Repairing HP9800 series calculators

Dr A. R. Duell[1]

March 2011

[1]ard@p850ug1.demon.co.uk

Contents

Chapter 1

Introduction

1.1 No Boardswapping!

There is only one way to repair a calculator (or any other electronic or mechanical device for that matter). That is to understand how it works and what it should be doing, to make measurements to see what actually *is* doing, then to work out what faults are causing the observed behaviour to differ from the correct behaviour and then (and only then) to correct those faults. Essentially random replacement of large parts of the machine (for example complete PCBs) will get you nowhere fast!

With that in mind, This manual will begin by describing how the HP9810, HP9820 and HP9830 calculators work. It won't meniton every logic gate, but hopefully enough detail will be give so that it's possible to follow the schrmatics nad understand exactly waht every seciton does. Further chapters will describe how to dismantle and inspect the machine, trace faults in the logic and perform the mechanical repairs that are almost always necessary.

For the purposes of the description, I will divide the machine into the following sections : The bit-serial, microcoded **Central Processor**; The **Memory System**, containing the ROM and RAM boards, the memory address and data registers and the associated control circuitry; The **Display** and its driver circuitry; The **Keyboard** and its encoder circuitry; The **Printer**, or in the case of the 9830 the printer interface; The **Storage Device** – either the magnetic card reader of the 9810 and 9820 or the cassette tape drive of the 9830; and finally the **Power Supply**

1.2 Sources of further information

In order to follow the descriptions, it is necessary to have the schematic diagrams for the machine, and data sheets on the ICs used. Therefore a TTL Data Book is very useful, but it needs to be a fairly old one to include some of the devices used.

Many of the ICs ins the actual machine will be marked with an HP 'house number' rather than the industry-standard code. A list of equivalents, such as the ones published in HP 'Bench Briefs' is almost essential when you come to do actual logic repairs. It is important to note that devices are removed from later lists (if no instrument supported by HP at that time uses the device, it will be removed), so you may have to check several such lists.

The HP service manuals are what I somewhat pejoratively call 'boardswapper guides' but they do contain a little useful information and are worth reading too. Finally, to fully understand the operation of the processor, is is useful to have a source listing of the microcode.

Chapter 2

CPU Theory of Operation

2.1 Introduction

All the HP9800 machines use the same CPU. It is a 16 bit bit-serial machine with 2 16-bit accumulators (A and B), a 16 bit program counter (P), and a 4 bit extension register (E). A 16 bit Q (Qualifier) registers is used to hold the current machine instruction and for some other internal purposes. The processor communicates with the machine's memory via the 16 bit memory address (M) and memory data (T) registers, which are considered to be part of the memory system and will not be described in this section. Although it is a bit-serial machine for binary operations, 4 bit BCD operations are possible between the T register and the A accumulator.

The processor is controlled by a 256 word microprogram stored in PROMs on the CPU control PCB. All HP9800 machines have the same microcode, and thus the same machine code instruction set.

2.2 Clock generator

In a bit serial machine, it is frequently the case that the same operation has to be performed several times on successive bits of the machine word. For example, to increment the value stored in a register, 1 is added to the LSB and then 0 is added to each of the remaining bits (Of course a carry bit is maintained between these bit additions).

(Those who already have some understanding of the machine may look at the microcode instructions at locations 0202 and 1603 where exactly this sequence is used to increment the program counter before fetching the next machine instruction)

There are thus 2 main clock signals in the HP9800. The **Bitclk** signal

drives the data shift registers and thus causes the current operation to be performed on the next bit of the work. The **μclk** signal drives the microcode program counter, causing the next microcode instruction to executed. The main purpose of the clock generator circuit is to produce m **Bitclk** pulses followed by a **μclk** pulse. then n **Bitclk** pulses, and so on, where the values of m and n are given by the current microinstructions. The **LdClk** signal which appears on some of the other PCBs is the same as **μclk**, the signals are linked together on the **CPU Clock PCB.**

2.2.1 ¡Master Clock

We begin on the **CPU Clock PCB**. The master clock is built round U5, and provides an 8MHz signal, **MClk**. There is an external clock input connector on the top edge of the PCB. Shorting together the 2 triangular pads on the solder side inhibits U5a and U5b, stopping the master clock, while an external clock signal (at TTL levels) can be fed in on the pad on the component side to allow the CPU to be run at other speeds, or single-stepped, for testing.

2.2.2 System Clock Generator

The central component in the system clock generator is a 4 bit counter, U5. This is loaded with the number of **Bitclk** pulses needed for the current microinstruction. Normally the 'Br' (Borrow) output is high, so U7a and U7b are both set, U12a and U13a pass master clock pulses through to **Bitclk** while U12b and U13b cause **μClk** to remain high. The counter is decremented on each rising edge of **MClk/**[1]. When the counter underflows, the Br output goes low during the low time of the clock input. Thus, on the next rising edge of **MClk/**, U7a is cleared, inhibiting the **Bitclk** signal and causing **μClk** to go low which changes the microcode program counter to point to the next microinstruction. Half an **MClk** cycle later, U7b is also cleared, loading the new bit count value into U14. After another half-cycle, U7a is set, re-enabling U12a (and thus **Bitclk**). Finally, after another half cycle, U7b is cleared, the system returns to the initial state with the new bit count value in U14.

Thus the total number of **Bitclk** pulses between **μClk** falling edges is one more than the value loaded into U14.

The **Bitclk** signal can be disabled by the IQN signal from the **CPU Control** PCB, thus allowing condition control of ALU operations.

[1] A '/' on the end of a signal name will be used to indicate an active-low or inverted signal.

2.3 CPU control and Microcode

2.3.1 Microcode Word

The CPU microcode consists of 256 28-bit words stored in 7 PROM chips on the **CPU Control** PCB. The microinstruction bits have the following meaning :

0...7 : Specify the next microinstruction address. Bits 0... 3 also specify the condition

8...11 : The value loaded into the clock divider counter and thus the number of **Bitclk** pulses to be generated in this microinstruction, minus 1.

12 : Read the currently-selected accumulator. If it is 0 and R=0, then do a Qμjmp (this will be explained later). If it is 0 and R=1 then begin an input/output operation

20, 13, 14 : Specify the operation to be performed by the ALU

15 : Apply the selected condition to the **Bitclk** signal, known as **IQN** in some HP manuals

16...18 : R field. This is mainly used to select one of the ALU inputs, but other functions are controlled here too. It will be described further below. Note that bit 16 is the MSB (and bit 18 the LSB) of this field.

19 : Use the currently-selected condition to control the next microcode address

21...23 : X field. This is used to specify the destination of the ALU output and various other operations. It will be described below. In this field, the bits should be taken in the order 22, 21, 23

24, 25 Specify the source for the second ALU input. This will be described as part of the memory circuitry

26 : Load the T register (Memory data) from the ALU output

27 : Load the M register (Memory address) from the ALU output

As an aside, the microcode PROM enable signal (**ROME/** is asserted by a 33Ω resistor on the **Data Path** PCB. The PROMs are normally always enabled, but could be disabled for factory testing.

2.3.2 Microcode Sequencer

The microcode address is provided by the 8 JK flip-flops in U8, U9, U16 and U17. The J and K inputs of each flip-flop are connected together, thus turning them into toggle flip-flops. If this combined input is low, then that flip-flop does nothing when clocked by the falling edge of μclk, whereas if it's high, the flip-flop will change state.

All these flip-flops are cleared (and thus all the microcode address bits set to '1') by the **Init** signal from the power supply. This starts the microcode at the correct address when the machine is turned on.

For the moment, ignore U15, and assume that it simply passes the signals on its '1' inputs through to its outputs. Thus the top 7 bits of the microcode address are directly controlled by the current microcode word address field. If a bit in this field is a '1', then the appropriate bit of the address is changed for the next microinstruction.

The lowest microcode address bit can be modified by a condition code (the **CC/** signal). If microcode bit 19 (μ**(19)**) is '0', then U11a is inhibited, so the output of U1d is forced high. In this case, the condition is not used, the change to address bit 0 is specified by μ**(7)** only. But if μ**(19)** is '1', U11a is enabled, so the **CC/** signal is passed on to U11b. Thus whether or not the lowest microcode address bit is changed depends on the state of the **CC/** signal.

Multiplexer U3 is the source of the **CC/** signal. It selects one of 16 conditions controlled by the value of the lowest 4bits of the current microinstruction. Ten of these conditions are bits in the Q register, The remaining 6 are the low bit of the program counter (8); the output of U4b (which indicates when the current machine instruction in the Q register is a memory reference instruction) (10); the binary (7) and decimal (14) carry flags; that an I/O operation is in progress (15); and finally that an interrupt has been received (12). (Numbers in parentheses indicate the input of U3).

Now consider U15, which is a 2-input multiplexer. Normally, as I suggested earlier, it causes the microcode address to be updated by the appropriate bits of the current microinstruction. But if the **QμJmp/** signal is asserted (low), then the top 4 bits of the microcode address are changed based on the value in the Q register. This is used as a multi-way microcode branch for decoding the current machine code instruction. The **QμJmp/** signal comes from U15b on the **Data Path** PCB and is asserted if microcode bit 12 is 0 and the R field contains 1 (as noted above).

Another simple microcode function uses U4a. This NAND gate makes the **IQN** signal low, inhibiting the **Bitclk** signal if the microinstruction bit 15 is high and the **CC/** signal is not asserted (that is, CC/ is high).

There is a very useful test connector on the top of the **CPU Control** PCB. It carries the microcode address bus, and many of the signals used to update the microcode address. Connecting a logic analyser to the microcode address bus will be suggested in a later article as a way of finding out just what the CPU is doing.

2.3.3 R and X field decoding

The R field is decoded by the 1-of-8 decoder U18 and the X field is likewise decoded by U10. The R field has the following meanings :

R=0 : Force '1' into ALU input

R=1 : Read P register (Program counter) into the ALU

R=2 : Shift T into the E register and E into the ALU

R=3 : Memory Write cycle

R=4 : Transfer ALU output to bit 6 of the Q register

R=5 : Read Q register into the ALU

R=6 : Memory Read cycle

R=7 : None of the above, used to force a '0' into the ALU input

While the X field has the following meanings :

X=0 : Shift ALU output into the Q register

X=1 : Set the accumulator selection flip-flop from bit 11 of the Q register and clear the decimal (BCD) carry flag

X=2 : Perform ALU decimal (BCD) operation

X=3 : Shift ALU into the E register and E into the ALU

X=4 : Invert the accumulator selection flip-flop

X=5 : Shift ALU into the P register

X=6 : Shift ALU into the currently-selected accumulator

X=7 : None of the above

This is one odd case. If X=3 and R=2, then the output of the ALU is shifted into the selected accumulator rather than E. E is read into ALU as expected, but the input to the E register is forced to 0.

2.3.4 Q Register

It is arguable whether the Q register should be covered here or in the date path section. For no better reason that it's located on the **CPU Control** PCB, I will describe it here. The Q register is a 16 bit shift register made from 4 4-bit devices. The control signals are not simply linked together (as would be conventional), so the operation of this register requires some explanation.

There are 3 operations that can be performed with the Q register (other than using its parallel outputs, e.g. as inputs to the microcode condition multiplexer) :

The Q register can be loaded from the ALU output. In this case, the X field contains 0, so **X(0)** is asserted (low). This enables the clock to all sections of the register via U4c and U11c, and all the mode inputs are low. Thus the register shifts one bit to towards the bit 0 end on each **Bitclk** cycle, and the output of the ALU **ALU(0)** is loaded into the most significant end of the register.

The Q register can be read into the ALU. The operation on the Q register itself is to rotate it one bit towards the bit 0 end on each **Bitclk** pulse. In this case, **R(5)** is asserted (low). The clock is enabled as before, but this time the mode input of the most significant device (U3) only is high. This causes this device to perform a parallel load operation, which owing to the connections to its inputs actually causes it to shift one bit towards the least significant end and load bit 0 into the most significant location. In other words the Q register rotates as required

Bit 6 (only) may be loaded from the ALU output. Now **R(4)** is asserted, so the output of U1e is high. The main clock is disabled, but U13 gets a load clock from the **LdClk** signal at the end of the microinstruction. Thus U13 parallel-loads. Three of its bits are simply copied back into itself, but the B input, corresponding to bit 6 of the Q register, is driven by the ALU output.

2.4 Data Path, Registers and ALU

2.4.1 ALU

The next section to consider mostly located on the **Data Path** PCB. The ALU itself consists of a pair of PROMs at locations U19 and U11. For binary (bit serial) operations, only U19 is used. Since in this case the X field of the microcode word does not contain 2, **X2** is low (due to U18c). This disables the higher-order T register inputs via U20a and U20d, and allows these inputs to the PROM to be driven by the ALU function select bits of the microinstruction. The bit-serial inputs to the ALU are **ALUSerIn** which

comes from a data selector in the memory system; and **Regser** fro a data selector on this PCB which will be explained shortly. The output of this ALU is **ALU(0)** which is fed to the various CPU registers.

For decimal (BCD) operations, both PROMs are used. **X2** is high, so U20a and U20d are enabled (there is no conflict with the microinstruction since for such microinstructions, μ**(13)** and μ**(14)** are both set to '1'. The low nybble of the A accumulator and that of the T register are fed into the ALU (the LSBs via the data selectors previously mentioned), the 4 it output is fed back to the A accumulator.

Flip-flop U12b acts as the carry flag for binary operations. It is enabled when μ**(20)** is '1' (ALU operations 4...7 are the ones which alter the carry flag). Similarly U12a is the BCD (decimal) carry flag, which is enabled when the X field contains 2. The carry flags can be tested as microcode conditions, and the appropriate one is fed back to the ALU circuit via U20b or U20c. One other feature that should be noted is that the binary carry flag can be directly set by the **SCF** signal via U13b. This signal comes from the input/output system and allows the I/O flag testing instructions to skip the next machine instruction if the I/O flag is in the appropriate state.

2.4.2 P Register

The processor registers are shift registers which can either be rotated (recirculated) for reading one bit at a time into the ALU, or which can be loaded by shifting the ALU output into the most significant end. The complexity of the register control circuitry depends on the register, so we will begin with the simplest one, the P register (or program counter).

The P register consists of a dual 8-bit shift register chip at location U10. The 2 sections are cascaded to form a 16 bit register. For reading, the R=1, so **R(1)** is low. The output of U18d is thus high, so U18b applies **Bitclk** to the register. The DS input of U10b is high (since the X field does not contain 5 for a simple read operation), so the data input to U10b comes from the output of U10a, the register recirculates. For writing, the X field contains 5. The clock signal is enabled as before but now the DS input of U10b is low, so the register is loaded from **ALU(0)**. Of course during a write operation, the current least significant bit of the register is available on the **Pout** signal, so the P register can be modified (for example incremented) without using an intermediate register.

2.4.3 Accumulator Control

There are 2 main accumulators, A and B, and in many cases they can be used interchangeably. In order to simplify the microcode (by allowing the same routines to be used for either accumulator), the accumulators are selected by U2 on the **CPU Control** PCB. This JK flip-flop is controlled in 2 ways. If X=4, then both J and K are set to 1 (via U1a) so the flip-flop toggles, selecting the other accumulator, whereas if the X=1, then U1f causes **Q(11)** to be applied to the J input and **Q(11)/** (from U1b) to go to the K input. Thus the flip-flop is set according to the 11th bit of the Q register. It turns out that in many cases a machine instruction which accesses one accumulator has a counterpart which accesses the other accumulator, and they differ only in the state of bit 11. Of course in other cases, the appropriate bit can be shifted into the correct position in the Q register before loading it into the selection flip-flop.

Back on the **Data Path** PC, the **Ld/** signal from U14a is asserted (low) when either accumulator is to be loaded from the ALU. This occurs when the X=6 (normal load to the accumulator), or R=2 and X=3 (the 'special case' involving the E register), or when data is being read from the input/output system.

2.4.4 B Accumulator

The B accumulator is similar in design to the P registers. It consists of a 16 bit shift register U9. For reading, μ**(12)** and **BEn/** are both low, so U16a enables the clock via U15c and U17a. The DS input of U9a is low, so the register recirculates. For writing, **Ld/** and **BEn/** are both low. U16b enables the clock input essentially as before, but since DS of U9a is now high, the register is loaded from the ALU output.

2.4.5 A Accumulator

The A accumulator is more complicated due to the fact that the low 4 bits can be read or written at once for BCD operations. It consists of a 4 bit register U4 cascaded to an 8 bit registers U6 cascaded to a further 4 bit register U5.

Binary read of A : μ**(12)** and **AEn/** are both low. The clock is enabled to all sections of the register by U16d and U15d. X is not 2, so X2 is low, as is the mode input of U5. The Mode input of U4 is also low, so the entire register recirculates. The data output is taken from **A(0)**

Binary write of A : **Ld/** and **AEn/** are both low, so the clock is enabled to all sections of the register via U16c and U15d. The mode input to U5 is low as before, but that to U4 is now high. Thus U4 performs a parallel load operation, but due to its connections, this is effectively a shift reading the output of the ALU into the most significant bit. The remainder of the register shifts as before. The overall effect is to shift the register one bit to the right and load the ALU output into the most significant bit.

BCD write of A . The clock to the upper sections of the register is disabled. Since X=2 and R≠0, the mode input of U5 is high via U8c. Thus U5 is parallel-loaded with the 4 bit output of the ALU, while the remainder of the register is unchanged.

2.4.6 E register

Although the E register consists of a single 4 bit shift register U1, it has the most complicated control circuitry of any of the processor registers. The R register is used when R=2, or X=3, or both and in all cases the clock to the E register is enabled by U7d and U8a. It is easiest to consider the 3 cases separately.

X=3, R≠2 : The mode input of U1 is high via U8b, so the register parallel-loads. Owing to its connections, the causes a right shift, loading the output of the ALU into the most significant bit.

X≠3, R=2 : The mode input is now low, so the register shifts right in the normal way. U14b is enabled, so the output of the T register (**T(0)**) is loaded into the E register

X=3, R=2 : Again the register shifts right as the mode input is low. U14b is now inhibited, so a 0 is shifted into the most significant bit of E.

2.4.7 The ALU input data selector

The input to the ALU, **RegSer** is selected by the AND-OR-Invert gate U3, expanded by U2. There are 6 possible inputs as follows : When R=0, U17d forces a 1 into the ALU input; When **BRd** is asserted by U16a, the output of the B accumulator is fed to the ALU; When R=1, the P register output is used; When **ARd** is asserted by U16d, the A accumulator is read into the ALU; when R=5, the Q register is read; and finally **ERd** from U7d causes the E register output to be passed to the ALU.

2.5 Input/Output System

Although HP refer the input/output controller as a 'processor' in their documentation, I think it is is more correct to refer to this section as a state machine. It transfers data between an accumulator and the I/O register. The latter provides a parallel input/output bus to connect to the various peripherals.

2.5.1 Input/Output controller

The input/output controller is located on the **CPU Clock** PCB. It is a state machine which decodes part of the machine instruction in the Q register and controls the I/O register and related circuitry. The microcode is not involved in decoding individual I/O instructions, the same microinstructions are executed for all such operations.

An I/O instruction has the I/O operation code in bits 5...8, while bits 0...4 provide 6 'select codes' (00000, 00001, 00010, 00100, 01000 and 10000) for various devices. Bit 9 is uses for some flag operations, the remaing bits are essentially the same for all I/O operations and are thus decoded by the microcode.

When no I/O operation is in progress (**I/OOp** is deasserted), the I/O operation bits (bits 5...8 in the Q register) are transfered to the I/O command register in U16 and U17 via the NAND gates U15d, U11b, U15c and U15b respectively by the **I/OCmdEn/** signal produced by U6d at the end of each microinstruction. The outputs of this register are decoded by U9 which is eanbled by **I/OOp/** but which is is also briefly disabled by **I/OCmdEn/** at the end of each microinstruction. Thus the outputs of this decoder are pulses. For example, a logical OR of outputs of this decoder (from U3a) is used to clock the I/O register, as descibed below.

An I/O operation begins with a microinstruction where μ**(12)** is '0', R=1 and **Q(10)** is '1'. This clears the JK flip-flop U1, thus asserting **I/OOp**. The I/O command decoder U9 is thus enabled. At this point, the processor microcode enters a 2 instruction loop, waiting for **I/OOp** to go low again, bringing **Cond(15)** low via U2c.

There are 2 types of I/O operations. Those which involve a data transfer, and those (such as flag operations) which do not. The output of U3a (**I/OClk** distinguishes between them, being high in the former case and low in the latter.

In the case of a non-transfer operation, the state machine behaves as follows. Since **I/OClk** is low, the output of U19a is high. Thus, when **I/OOp** goes high, the output of u15a goes low, disabling the NAND gates feeding

the I/O command register. When **I/OCmdEn/** goes high, the command registers is loaded with all '1's. This causes **I/OCmd(15)** to be asserted (low) which then clears the I/O Operation flip-flop, thus ending the microcode loop as described above.

For data transfer operations, the number of bits to transfer (minus 1) is loaded into the 4 bit counter U4. Since **I/OClk** is now '1'. while the counter has not underflowed, U19a's output is 0, so the output of U15a is kept high. The I/O command register does not change.

When U4 underflows, **I/OBr** goes low, so the output of U19a goes high and that of U15a goes low. The input gates to the I/O command register are inhibited as before. U11a is enabled, forcing the D input of U17a low. The command registers is loaded with '1101', and this '**I/OCmd(13)**' state is used for any flag operations at the end of the data transfer. **I/OClk** is now low, so on the next clock pulse, the I/O command registers is loaded with '1111' and the I/O operation terminates as before.

2.5.2 Input/Output register

The input/output register is a 16 bit shift register located on the **I/O Interface** PCB. For communication with external peripherals, it provides 8 parallel data outputs (**DO(0)...DO(7)**), 4 status outputs (**SO(0)...SO(3)**) and 4 control outputs (**CO(0)...CO(3)**). The last are used for for a 4 bit peripheral address. However, for communication with internal devices such as the display, the register can be used to provide an arbitrary 16 bit word. On input operations it accepts 8 data bits from the peripheral (**DI(0)...DI(7)**) and 4 status inputs (**SI(0)...SI(3)**).

In I/O data transfer instructions, one bit is transferred for each microinstruction executed in the I/O loop. Although, for timing reasons, each microinstruction is 2 clock cycles long, every alternate **Bitclk** pulse is inhibited by U6c on the **CPU Clock** PCB. Thus only one **Bitclk** pulse is produced for each instruction. The microinstructions also cause the ALU to perform an inclusive OR operation

An output operation is performed when **I/OCmd(12)** is asserted. On the **CPU Clock** PCB, the μ**(12)** is forced low by U19d and U11c, causing the selected accumulator to be read. The data selector on the memory timing board applies logical 0s to the **ALUSerIn** input, so the data is passed through the ALU unchanged and latched in U13b on the **I/O Interface** PCB by the **Outclk** signal from U8c on the **CPU Clock** PCB.. The output of U13b is then transfered to the I/O register by the **I/OClk** signal. Depending on the state of **I/OSize**, either the entire 16 bit register or only are used.

There are 2 types of input operation. The simple input (corresponding to **I/OCmd(10** being asserted) transfers the I/O register to the selected accumulator, the more complex one (when **I/OCmd(14)** is asserted) inclusive-ORs the accumulator with the I/O register. The operation is very similar in the 2 cases. The parallel data from the peripheral has previously been loaded into the I/O register by the **I/ORd** signal and monostable U8. The I/O register is clocked as before, the **InEn** signal is asserted by U2d and U18b on the **CPU Clock** PCB. This causes the data selector in the memory system to feed the least significant bit of the I/O register (inverted by U20d on the **I/O Interface** PCB, to compensate for the inversion in the data selector) into the ALU. The date is thus read into the selected accumulator.

The only difference between the 2 types of input instructions is that for the second one, the accumulator is also enabled for reading by U19d and U11c on the **CPU Clock** PCB. This causes the data to be inclusive-ORed with the contents of the accumulator, rather than with 0.

2.5.3 I/O Strobes and Flags

The **I/O Interface** PCB generates strobe signals for th various peripherals. Since the logic involved is fairly simple, to save space only the display strobe (**DispStb/**) will be described here.

The display strobe is generated by the SR flip-flop consisting of the cross-coupled NAND gates U2c and U7a. The strobe is set by an end-of-data operation (**I/OCmd(13)** is asserted) when the select code is 01000 (**Q(3)** is high). It is cleared by an end-of-command operation (**I/OStb** is high) with the same select code).

Similarly, many devices have status flag bits, such as the card reader status flip-flop U4c and U4d. All the peripheral flags are combined by the AND-OR-Invert gate U9 (controlled by the device select code bits) and fed to the **I/OFlg** line which goes to the **CPU Clock** PCB. A skip-on-flag-set instruction, corresponding to **(I/OCmd(5)** being asserted) will set the CPU carry flag thus skipping the next instruction) if the associated flag is set (and thus **I/OFlg** is low) via U10d, U10c and U19c. The skip-on-flag-clear instruction (when **I/OCmd(1)** is asserted) does the same thing if **I/OFlg** is high via U19b and U19c.

2.5.4 Interrupt

The **KeyDn/** signal is really an interrupt input to the processor. It can be pulled low by any peripheral device whereupon the output of U3f on the **I/O Interface** PCB goes high. This brings **Cond(12)** low via U4a, and

this condition is tested by the microcode near the start of the fetch-execute cycle.

The interrupt handler microcode routine executes an instruction where μ**(12)** is low, R=1 and bit 10 of the Q register is 0. This does not start an I/O operation because **Q(10)** being low prevents U1 on the **CPU Clock** PCB from being cleared, Instead U3c asserts the **KeyAck/** signal which sets the SR flip-flop U2a,U2b on the **I/O Interface** PCB. Thus **Keylatch** goes low, inhibiting U1b and U4a, and deasserting **Cond(12)**. When the device deasserts its interrupt output, U2a, U2b is cleared and the processor is ready for the next interrupt.

2.6 A little of the Microcode

I do not intend to cover much of the microcode – those who are interested in how machine instructions are executed should spend some time with the microcode listings. However, since one way of determining what the CPU is actually doing is to examine the microcode address on the test connector with the logic analyser, I will give a few important addresses.

It is conventional to give the microcode addresses in 'split octal' format. The 8 bit address is split into 2 4 bit sections, each is separately converted to a 2 digit octal number and the these are concatenated. In other words, a microcode address consists of 4 digits. The first, either 0 or 1, gives the value of bit 7. The next, any octal digit, gives bits 6...4. The next, again either 0 or 1, gives bit 3, and the last digit gives bits 2...0.

The fetch-execute loop begins at 0202 which, together with 1603, increments the program counter. At 1612, the next machine instruction is read into the T register, and at 0616 it's transfered to Q. The interrupt signal is tested here too, The interrupt handler begins at 1602, while the normal instruction decode begins at 1213 by setting the accumulator selecting flip-flop and determines whether this is a memory-reference instruction

The other important routine for hardware debugging is the I/O loop at 0512 and 1207. The main processor stays in this loop while the I/O state machine is in operation. Should there be a failure with the latter, the processor may end up stuck in this loop. Of course tracing the microcode address will reveal this.

Chapter 3

Memory System Theory of Operation

3.1 Introduction

Although the differences between the memory systems in the 3 models of clacualtor mean that the systems will be described separately, there are genearl features common to all of them which will be described first.

The memory system consistrs of the ROMs (both those containing the standard calculator firmware and those in optional extension modules), the RAM, and the memroy cotnrol circuitry. The latter contains a pair of 16 bit shift registers, the memory address (or 'M') register is serially loaded with the current memroy address by the processor. Its parallel output select a particular location in the ROM or RAM. The memory data (or 'T') register provides an interface betwee nthe 16 bit parallel memory data bus and the bit-serial processor. For read operations, the contents of the selected memory location are parallel-loded into the memroy data register and trasnfered serially to the processor. For write operations, the memroy data register is loaded serially by the processor and then its contents are transfered to the addressed RAM location.

The parallel outputs of the M register address a 32K word address space (the MSB, **M(15)**, which correspods to the indireciton bit of a procesosr address, is used to trigger an eaxternal DMA device in the HP9820 and HP9830 machines, and is not regarded as part of the memory address). This single address space contains the ROM and RAM devices.

The CPU microocde word bits 24 and 25, known as the 'S field' (with bit 24 as the MSB) are decoded by the memory control system. This field controls the bit-serial input to the second ALU input (signal **ALUSerIn**)

and has the following meanings :

S=0 : Force 0 into the ALU

S=1 : Read the M register into the ALU

S=2 : Read the T register into the ALU

S=3 : Force 1 into the ALU

The ROM devices used are HP-designed 512*8 bit mask-programmed ROMs. They are somwahat unconventional, in that they are internally designed as a pair of 256*8 ROMS[1]. Each secion of the ROM is eanabled by a separate active-low input, these are conventionally linked to gate forms of address line **A8** and its inverse, so that when the ROM is accessed one (and only one) of the sections is eanabled. Another unconventional feature of these early MOS ROMs is that the enable inputs have to be taken to +12V votlage levels, rather than the (5V) TTL-compatible levels on all the other inputs and outputs.

This series of machines was one of the first major commercial applications for the Intel 1103 1K bit RRAM. These PMOS devices run from -16V supplies with a +3V bias supply, and have 16V votlage swings on the logic inputs and outputs. In the HP9800 series, the power supply pins are re-labelled as +16V and +19V (this makes no electrical differenec, you can connect th ecommon lead of your voltmeter anywhere), but of course bufferes are still needed to convert between the TTL level logic signals used elsewhere in the machine and the 16V ones used by the RAMs.

One further comment is that the processor requires ROM starting at locaiton 0 to provide the first instruciton to be executed at power-on, the 'counter constant' at locaiton 1 (this is a special bit pattern, (043060_8) used by the floating point microcode routines. However, the memory area from 1400_8 to 1777_8 must contain RAM to store the floating point data, the ssytem stack point, etc. This is no problem on the HP9810, which simply fils the first 512 words with a pair of ROMs and the next 512 words with a block of RAM but it slightly complicates the address decoder in the other calculators.

Physicaly, the memory system of the HP9810 and HP9820 machines is contained in a memtal boc plugged into the main backplane, Inside the box are the ROM and RAM PCBs and 3 boards of memory control circuity (**Memory Address**, **Memory Data**, and Memory Timing PCBs. The

[1]The organisation of the device and the data it contains are determined by the final metalisation layer on the chip. 256*16 bit ROMs using the same underlying device are found in other HP products of the period.

memory ssytem of the HP9830 consists of PCBs directly plugged into the main backplane, these consist of ROM and RAM PCBs, 2 boards (**Memory Addres** and **Memory Data**) contianing the memory control circutiry, and the **Ext ROM Seleector** PCB.

3.2 The HP9810 Memory System

The memory system of the HP9810 is unconventional in that a 1K*1 bit RAM chip appears as a 512*2 bit device. The 16-bit wide system RAM uses only 8 chips, for example.

3.2.1 M register and address decoder

The M register is a 16 bit shift register formed by cascading U2, U3, U8 and U1 on the **Memory Address** PCB. To load it from the ALU output, microcode bit μ**(27)** is low¿ This enables U9d, which passes the **Bitclk** signal to the shift register. The mode input is low, so the register shifts right on each **Bitclk** pulse, and the ALU output is shifted into the most significant end.

To read the register, the S field contains 1. This is decoded by U16c on the **Memory Timing** PCB, and the **RotEn** signal is high. Back on the Memory Address PCB, U9c is enabled, and for a simple read, the mode input of the shift register is high. The register thus parallel-loads on each **Bitclk** pulse, but owing to its connections, this results in a rotation to the right.. The LSB of the register is transfered to the ALU via the first gate of the AND-OR-Invert gate U17 on the **Memory Timing** PCB, enabled by the **RotEn** signal.

The 2 operations can occur simultaneously. In this case, the register performs a serial load operation, and the LSB is transfered to the ALU via the AND-OR-Invert gate.

The parallel outputs of the M register are decoded to provide chip-select signals for the ROM and RAM devices. The output of U9a is asserted (high) when the high nybble of the address is X000. This enables the decoder U17 via U11a when the **MemEn** signal is high. This signal, from U11b, is asserted if the R field contains 3 or 6, corresponding to memory read or write operations. Similarly, U10 is enabled via U9b and U11d when the high nybble of the address is X001. These decoders thus divide the lowest 8K words of the processor address space into sixteen 512-word blocks, which is the size of either a pair of ROMs or a set of RAMs.

The ROM selection outputs from these decoders are converted to 12V levels (as required by the ROM chips) using open-collector inverter and transistor circuit such as the one built round U15f. When the chip select line is not asserted, the input to U15f is high, so the output, and the select input to the ROM are low. The transistor is cut off. When the input to U15f goes low, its output is pulled high by the 4k7 resistor, the transistor is turned on, bringing the ROM select line to +12V, enabling that ROM

In a similar way, the RAM selection outputs are converted to 16V levels. For example, the **RawRAMCS(0)** signal from the decoder is logically ORed with the **Rfsh/** signal by U11c, the output of this gate will be high as will the output of U12b if the **RAMStb** signal from the memory timing PCB is high. The output of U12b is inverted and converted to 16V levels by U18e and a complementary pair of emitter-followers.

The **RAMEn** signal, from the NAND gate U4, is asserted if any of the RAM areas are selected.

The lower 9 outputs of the M register are used to select a location within a memory device. In the case of the ROM, the lowest 8 bits are simply buffered, for example by U7c, these NAND gates being enabled by the **MemEn** signal, delayed by U5a and an RC timing network. Bit 8 is used to enable one or other half of the ROM as described in the introduction, these 2 signals are produced by U13a and U13b, enabled by U14b, which occurs during the later part of a memory cycle (**ROMEn/** asserted) which doesn't address a RAM block.

The RAM addressing is slightly more complicated since, being dynamic RAMs, a refresh address has to be generated too. The top 5 address lines are converted to 16V logic levels by circuits such as the one based round U20f and sent to the RAM board (this circuit, of course, operates in the same manner as the ROM selection buffers). The low nybble of the M register is fed to the **Memory Timing** PCB, where it enters the AND-OR-Invert gates U14 and U15. These act as multiplexers, switching the RAM address buffer inputs between the output of the M register and the output of the refresh address counter, U13. RAM address input 4 (**RAMAddr(4)**) is used to select between the 2 half-width locations storing a word, and will be described below.

3.2.2 T Register

The 'T', or Memory Data, register is located on the **Memory Data** PCB, and consists of the four 4-bit registers at locations U1, U6, U11, and U14 cascaded in the conventional way. The serial input/output facilities of this register are used to communicate with the bit-serial processor, the parallel

signals are used to communicate with the memory device data bus.

To load the T register from the processor, microcode bit μ**(26)) is low. This causes the output of U21a on the memory Timing PCB to go high, enabling U11c and passing the (Bitclk** signal to the T register. The top gate of the AND-OR gate U22 on the **Memory Timing** PCB is enabled, so the output of the ALU is gated to **TRegSin** and thus loaded into the register.

To read the T register into the processor, the S field contains 2. This causes the output of U16d on the **Memory Timing** PCB to go high, which enables the T register clock via U21a and U11c as before. Now the second gate in U22 is enabled, so the LSB of the T register (**T(0)**) is gated back to the serial input, the register rotates. The second gate in the **ALUSerIn** data selector, U17, is enabled so the LSB of the T register is transfered to the ALU input.

If both operations occur simultaneously, then only the first gate of U22 is enabled so the T register is loaded from the ALU as before, but the second gate of U17 is also enabled, so the LSB of the T register is transfered to the ALU input.

As an aside, the third gate in U17 is used for force a '1' into the ALU input when the S field contains 3 and the fourth gate feeds the serial output of the I/O register on the **I/O Interface** PCB into the ALU when enabled by the I/O control circuitry during input instructions.

The parallel interface between the T register and memory will be described below.

3.2.3 Memory timing and control

The memory timing circuit is based around the 4-bit counter U5 on the **Memory Timing** PCB. The outputs of this counter are decoded by U10, giving the **State(n)** signals. Normally this counter is held reset by U4c, since both **State(0)** and the output of U12b are low. A memory cycle is stated by the **MemEn** signal going high. This forces to the output of U12c high and that of U12b high. The reset input of the counter thus goes low, and the counter starts counting, the **State(n)** lines going low in sequence. The output of U12b goes low again when the memory accessing microinstruction ends after 12 clock cycles, but the output of U4c remains low until the **State(0)** signal goes low again. The system is now back in the initial state, ready for another memory cycle.

There are 3 types of memory access operation :

ROM Read

In state 4, and all subsequent states, the output of U21d goes high, asserting **ROMEn/** via U23c and bringing **ROMCE** from 12V to 0V. The ROM address lines are enabled as described above. Since this is a read operation, **R(6)** is low, so **TMode** is driven high by U16b, setting the T register for parallel loading. The T register is loaded by the **TLd** signal, in state 5 (which is essentially a dummy operation) and again in state 11, due to U3b, U11d and U11b. Since **R(3)** is high. **THold** is forced low by U16a and since U2a is never set, **TShift** is also kept low.

On the **Memory Data** PCB, the **RAMEn** signal is low, so the RAM Read buffers are disabled. Thus (considering bit 15), the output of U16C is high, the ROM data passes through U2b into the T register. In the case of bit 14, both U5c and U5d are disabled, their outputs are '1', so U2a passes the ROM data into the T register. The other bits follow the same pattern.

RAM Read

The RAM read cycle has to be considered one state at a time (All ICs are on the **Memory Timing** PCB unless otherwise specified) :

State 2 : U8a is cleared by U3d, thus asserting **RAMStb** and enabling the selected RAM via the gates on the **Memory Address** PCB. U8b is directly set, so the **Precharge** input to the RAM goes high.

State 5 : U8a is set, deasserting **RAMStb**. **TLd** is asserted via U3b U11d and U111b, the RAM data outputs are converted to TTL levels on the Memory Data PCB (e.g by U18b) and transfered to the odd-numbered bits of the T register, the **ROMD** lines all being high since no ROMs are enabled..

State 6 : U8b is cleared by U3c, thus the **Precharge** signal goes low. U9b is toggled via U23f, thus **RAMAddr(4)** goes high, selecting the location containing the other half of the word.

State 8 : U2a is set via U11a, thus asserting **TShift** (which remains asserted for the rest of the cycle).

State 9 : U8a is again set, this is a repeat of State 2 to read the second half of the word.

State 11 : The **TLd** signal is again asserted by U3b, U11d and U11b. The RAM outputs are transfered to the odd-numbered bits of the T

register as in state 6, but since **TShift** is asserted, the transfer gates, for example U5c on the **Memory Data** PCB are enabled and the existing contents of the odd bits of the T register are transfered to the even bits.

State 12 : U8a is set, disabling the RAM select signal.

Sate 15 and end of cycle : All flip-flops are returned to their initial states, the state counter is halted.

Thus, at the end of the cycle, The 2 register contains the contents of the first location read (**RAMAddr(4)**=0) in its even-numbered bits and the contents of the second location read (**RAMAddr(4)**=1) in its odd-numbered bits.

RAM Write

The operation of the **RAMStb** and **Precharge** signals is identical to a read cycle. U16a brings **THold** high during states 0-7, since **R(3)** is low.

State 2 : RAM selected and Precharge brought high as before

State 4 : Since **THold** is high and **TShift** is low, the even-numbered bits of the T register are applied to the RAM data inputs via, e.g., U5d and U10d on the **Memory Data** PCB. U9a on the **Memory Timing** PCB is cleared for one cycle, pulsing the **R/W/** signal low and writing this data into the RAM.

State 5 ; U8a is set, removing the RAM select signal

State 6 : U8b is cleared, removing the **Precharge** signal. U9b is set to select the second RAM location to be used (as above)

State 8 : U16a is now disabled, so **THold** goes low. U2a is set via U11a, so **TShift** goes high/

State 9 : Start the second RAM cycle, as in State 2

State 11 : Since **TShift** is high and **THold** low, the odd-numbered bits of the T register are gated to the RAM inputs (e.g. via U5c and U10c on the **Memory Data** PCB). U9a is cleared for one cycle (as in State 4) thus writing the odd-numbered data bits to the RAM.

State 12 : Set U8a to disable the RAM select signal as before

State 15 and end of cycle : All flip-flops are returned to their initial conditions ready for another cycle.

Thus, at the end of the cycle, the even numbered bits of the T register have been stored in the first RAM location (**RAMAddr(4)**=0) and the odd-numbered bits of T in the second RAM location (**RAMAddr(4)**=1), as required for them to be correctly read back by a RAM Read cycle.

3.2.4 RAM Refreshing

The dynamic RAM ICs used in the HP9810 need to be refreshed and this is performed by circuitry on the **Memory Timing** PCB.

The monostable U1 sets the time between refresh cycles. When it times out, it causes U6a to be set, thus removing the rest signal from U6b. At the end of the next microinstruction, the **RfshSt** signal is asserted via U12d and U4d, so U6b is set. Thus the output of U18a goes high, asserting the **Rfsh** signal and inhibiting the microcode clock via U20d.

When **Rfsh/** is low, the reset input of the state counter is also low, thus this counter free-runs. The memory timing circuitry performs dummy RAM ready cycles. U9a is held set, so the **R/W/** input to the RAM can never go low and writing is thus impossible. U16b is inhibited, so no data is transfered to the T register. RAM address bit **RAMAddr(4)** is provided by U9b as usual.

The AND-OR-Invert gates U14 ad U15 switch the low 4 RAM address lines from the outputs of the M register to the outputs of the 4-bit counter U13. This is incremented at the end of every complete refresh memory cycle by U12a. When all 32 addresses have been accessed, as required by the RAM's specifications. (a total of 16 cycles), **RfshCy** goes low. This triggers U1, deasserting the **Rfsh** signal. The machine then continues running until U1 times out again and the RAM is refreshed once more.

3.2.5 Memory PCBs

The ROM and RAM PCBs, along with the plug-in ROM modules simply contain the appropriate memory devices and little more needs to be said about them. One minor curiosity is that the user program in the HP9810 is stored as a sequence of 6-bit keycodes, and thus the program storage RAM is logically 6 bits wide. Due to the fact that the physical RAM width is half the logical RAM width, the program RAM is expanded in multiples of 3 chips. I suspect this is the only machine where this is the case.

3.2.6 Memory system test connector

There is a test connector on the top edge of the memory box backplane in both the HP9810 and HP9820 machines. It carries the parallel outputs from the M and T registers and can be used to check the proper functioning of these registers or to trace machine code programs. The pinout of this connector is the same in the 2 machines.

3.3 The HP9820 Memory System

The HP9820 uses 16-bit wide memory throughout, both ROM and RAM. This simplifies the design of the T register and memory timing circuits, but causes additional complexity in the address decoder.

3.3.1 M register and address decoder

The M register itself is located on the **Memory Address** PCB, and is very similar to that in the HP9810. It is a 16 bit shift register formed by cascading U4, U3, U9 and U2. It is serially loaded from the ALU output when the microcode bit μ**(27)** is low, This enables the **SRClk** signal via U10a and U122. The mode inputs of the shift registers are low, so the register shift right and the output of the ALU is loaded into the MSB.

When the S field contains 1, the output of U6b on the **Memory Timing** PCB goes high. This enables U12a on the **Memory Address** PCB. The M register rotates as described in the case of the HP9810. The LSB of the M register is transfered to the ALU via the second gate of the AND-OR-invert gate data selector U25 on the **Memory Timing** PCB, enabled by U6b. As in the HP9810, simultaneous reading and writing are possible, and operate in the same way as in that machine.

The parallel outputs of the M register are decoded and used to enable the various memory devices. The output of U5c on the **Memory Address** PCB goes high when the address is in the first 4K of the address space, and thus decoder U14, enabled by U23d, decodes this part of the space into 512-word blocks. Of course **Memcycle.** the output of U23d is high for a memory cycle when the R field contains either 3 or 6. The output of U5a is high for addresses of the form $00xxxxxxxxxxxx_2$ or $0x000xxxxxxxxx_2$. Since in this part of the address decoder, the output of U5a is ANDed with **M(12)** by U12, the second group of addresses is masked out here, and the decoder U15 decodes the second 4K part of the address space.

ROM addressing

There are 14 ROM blocks in the HP9820 (8 for the system ROMs and 2 in each of the 3 plug-in modules). Thirteen of these blocks are selected by outputs of these decoders, which are buffered to the required 12V levels by inverter and transistor circuits, such as that built round U19b on the **Memory Address** PCB. The operation of this circuit has been described as part of the HP9810 memory system.

ROM 1 is the exception to this. It appears in the first half of blocks 1 and 14. When either block is selected, the output of U16d goes high. If **M(8)** is also low (signifying the first half of a 512-word block), the output of U10f will also be high. This enables ROM 1 via U1d and U19e.

The lowest address lines are passed through e.g. U1b (enabled whenever a memory cycle is occurring) and fed to the ROM devices. Address line 8 is a little more complicated. Normally, **RawCS(14)** is high, thus the output of U16 is the same as **M(8)**. This is used by U13a and U13b to enable the appropriate half of the ROM when **ROMHAddrEn** is high. This comes from U17b, and occurs during a memory cycle which does not access RAM (and therefore accesses ROM). When **RawCS(14)** is low – when the second half of ROM 1 is accessed, then ROM address input 8 is forced high by U16a, thus selecting the second half of ROM 1 (of course, during accesses to the first half of block 1, **M(8)** is low anyway, thus selecting the first half of ROM 1 here).

The output of U17d (**StatEN**) goes high when the last word of the ROM in block 1 is read. This word is modified if an extension RAM PCB is fitted to the machine by logic on the **Memory Data** PCB.

RAM addressing

The standard 1K words of RAM appears in the second half of block 1, the second half of block 2 and all of block 15. The output of U16d is high whenever blocks 1 or 14 are accessed, and thus the output of U18c is high whenever the second half of either of these blocks is accessed. The output of U17a is clearly high when block 15 is accessed. These are logically ORed by U17c, the output of which goes low to enable the standard RAM.

The expansion RAM (also 1K words in size) is enabled for addresses of the form $01000xxxxxxxxx_2$ by U12b and U23c. When either RAM area is accessed, the output of U16c goes high, enabling the RAM output buffers on the **Memory Data** PCB and disabling ROM access via U17b.

The outputs of the RAM decoders are logically ORed with the **Rfsh** signal (so that all RAM is enabled on a refresh cycle) by U11c and U11d,

and then ANDed with the memory timing signal **RAMCSEn** by U18d and U18a. They are then converted to 16V levels and fed to the RAM chips.

The address lines to the RAM chips Fit into 3 groups. **M(0)...M(4)** are fed to AND-OR-Invert gates on the **Memory Timing** PCB (e.g U22a) which elect between the M register outputs and the refresh address counter. The outputs of those are buffered to 16V levels (e.g. by U17a) and fed to the RAMs, as described in the HP9810 memory system.

M(5)...M(7) are simply converted to 16V levels on the **Memory Address** PCB (e.g.by the circuit around U21e) and fed to the RAMs.

The highest 2 RAM address line circuitry is also on the **Memory Address** PCB but is made more complicated by the fact that the standard RAM is split between 3 blocks. When block 1 is accessed, U16b and U11b are inhibited, so the outputs of both are high. When the second half of block 14 is accessed, the output of U16b goes low, that of U11b is high (by virtue of the fact that **M(9)** is low in block 14). And throughout block 15, the output of U16b is the inverse of **M(8)** while U11b's output goes low (since **M(9)** is high as is **RawCS(1)**). Thus a different 256 words of RAM are accessed in each case. When the expansion RAM is accessed, the outputs of U16b and U11b are simply the inverses of **M(8)** and **M(9)**.

3.3.2 T Register

The T register is a 16 bit shift register comprising U2, U5, U3 and U4 on the **Memory Data** PCB. As in all HP9800 machines, it communicates serially with the processor while the parallel inputs and outputs are used for the parallel memory data bus.

To load the T register from the processor, μ**(26)** is low, This enables the **Bitclk** signal to the R register via U7c, U11b and U13a on the **Memory Timing** PCB. The ALU output is passed to the MSB of the T register via the third gate in the AND-OR data selector U18 on the **Memory Timing** PCB, so data is transfered from the ALU to the T register.

To read the T register into the ALU, the S field contains 2. This causes the output of U9d on the **Memory Timing** PCB to go high, enabling the T register clock as before. The second gate in U18 is enabled, so the T register recirculates (the LSB output is fed back to the serial input). The fourth gate in the ALU data selector U25 is enabled so the serial output of the T register is read into the ALU.

If both operations are requested at the same time, the T register is read into the ALU as normal, but the third (not the second) gate of U18 is enabled, so the T register is loaded from the ALU output.

The remaining gates in U18 allow the T register to be loaded from an external DAM device (fourth gate) or to recirculate when being read by a DMA device (first gate). Little more will be said about DMA devices here, since I have never seen one and thus have no precise details on how they operate.

There are 2 remaining gates in U25. The first gate is used to force a '1' into the ALU when the S field contains 3. The third gate transfers the I/O register contents to the ALU under the command of the input/output system. These operate identically to their counterparts in the HP9810.

The parallel outputs of the T register are converted to the RAM's 16V levels, e.g. by the circuit containing U8c, and fed to the data inputs of the RAMs.

The data outputs from the RAMs are converted to TTL levels by the comparator circuits such as U14a, which are enabled when the RAM is accessed. The outputs are then inverted and logically ORed with the ROM data outputs, for example by U16, and fed into the parallel inputs of the T register. Since ROM and RAM are never both enabled at the same time, this simply passes the data from an enabled memory device, unchanged, to the T register.

There is one exception to this. **TIn(10)** will be forced high by U10 when the last word of the ROM in block 1 is read (that is, when **StatEn** is asserted) and if an expansion RAM board is fitted (when U11a's output is high since its input is connected to ground by track on this extension RAM board). Thus the machine can tell if the board is present.

The timing of the parallel load will be described in the next section.

3.3.3 Memory timing and control

The memory timing circuit is again based around a 4 bit counter, U1 on the **Memory Timing** PCB, the outputs of which are decoded by U5. Normally the counter is held reset by U6a, but when a memory cycle occurs, the output of U7d goes high, releasing the reset condition. The counter thus counts until **State(0)** goes low at the end of the cycle. The **ROMEn/** signal is asserted by U11d and U21f when the state counter gets to state 4, and remains asserted for the rest of the cycle.

There is essentially only one type of memory cycle in the HP9820, which occurs as follows :

State 1 : U16b is set, removing the reset signal from U15a and allowing the **Precharge** signal to be asserted later.

State 2 : Start of the dummy RAM cycle, used only during refresh. U15b is cleared, asserting **RAMCSEn**, setting U15a via U9b and thus asserting **Precharge** to the RAM

State 5 : Set U15b, deasserting **RAMCSEn** and removing the set signal from U15a. End of the first RAM cycle.

State 6 : Clear U15a, deasserting **Precharge**. Toggle U4a. This is a remnant of the half-width RAM addressing of the HP9810, and is used here only to supply a bit of the refresh address and thus refresh two RAM locations in a single refresh cycle.

State 9 : Clear U15b, starting the second RAM cycle as in state 2.

State 11 : On RAM write cycle, U16a is toggled low for 1 clock cycle (the output of U22a is high), so **RAMWr/** is asserted, writing the data to RAM. For read cycles, **TMode** is high (since U12d's output is high), and thus **TLd** pulses high in this state. This parallel-loads the T register with the contents of the selected ROM or RAM location.

State 12 : End of the second RAM cycle, as state 5.

State 15 and end of cycle : All flip-flops are returned to their initial states, ready for another cycle, the counter is halted.

Thus at the end of the cycle, the appropriate transfer between the T register and memory has occurred.

3.3.4 RAM Refresh

The Dynamic RAM refresh circuit is located on the **Memory Timing** PCB. A discrete transistor astable multivibrator releases normally holds U4b reset. When a refresh cycle is to occur, this reset is released, and at the end of the next microinstruction, U14b is set, asserting the **Rfsh** signal and inhibiting the microcode clock via U19c. When **Rfsh** is asserted, the reset input of U1 is held low via U7d and U6a, so the memory state counter runs. As described above, 2 memory cycles are performed for each complete cycle of this counter. **RAMWr** is inhibited since U16a is held in the set state and T register loads are inhibits via U12d and U8a. Thus no actual data transfers occur in these cycles.

The AND-OR-invert data selectors such as U23 switch the low 5 RAM address lines from the outputs of the M register to the outputs of the refresh address counter on the **Memory Timing** PCB. The LSB of the refresh

address is provided by U4a, which as described above is toggled during the state counter cycle. The remaing 4 bits are provided by a 4 bit counter, U20, which is incremented at the end of each state counter cycle. Thus all 32 refresh addresses are generated in sequence. When all locations have been refreshed, U14a is set via U21a, inhibiting U9a, deasserting **Rfsh** and enabling the microcode clock/ U14a is cleared when U14b is cleared by the multivibrator, ready for another refresh cycle.

3.3.5 DMA flip-flop

Since I do not own a DMA peripheral, I am unable to describe DMA cycles in detail. However, since this flip-flop disables the system microcode clock, it will be described.

The DMA control flip-flop is U4b on the **Memory Timing** PCB. When a memory cycle is not occuring (**Memcycle** is low), it is held reset. Therefore, when a memory cycle begins, both inputs of U13c are high and thus **DisμClk(2)** goes low, This would inhibit **μClk** via U12b on the **Clock** PCB.

However, if no DMA operations are occuring, **EMB/** is high. Therefore, when U4a is set during the memory cycle, the output of U9c goes high amd U4b is set. This re-enables the microcode clock before the start of the next microinstrucion, in fact this flip-flop has no real effect on the machine. But during DMA operations, will rempain clear, preventing further microinstructions from being executed untl the DMA operation has completed.

3.3.6 Memory PCBs

As in the HP9810, the HP9820 memory PCBs contain the appropriate memory devices only. The standard and expansion RAM PCBs are identical and simply contain sixteen 1103 DRAM devices. The 2 ROM PCBs are electrically identical both to each other and to the HP9810 ROM PCB. The boards differ, of course, in the programming for the ROM devices fitted, and also in the part numbers, both for the complete PCB and the ROMs themselves that are etched on the board. The optional ROM modules again are electrically identical to those in the HP9810, but of course with different ROMs.

3.3.7 Memory system test connector

The test connector on the top edge of the memory box is identical in connections and function to that on the HP9810 memory box.

3.4 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.

3.4.1 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.

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

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.

3.4.2 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.

3.4.3 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.

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.

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.

3.4.4 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.

3.4.5 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.

3.4.6 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.

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.

3.4.7 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.

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.

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.

3.4.8 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.

3.4.9 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.

Chapter 4

Display Theory of Operation

The HP9810, HP9820 and HP9830 obvious have very different display systems. The HP9810 has 3 rows of 15 digits, the HP9820 has a 16 character dot-matrix display and the HP9830 has a 32 character dot-matrix display. However, in all 3 cases, the display is controlled by the 16 bits of the I/O register. The system firmware loads a value into the this register and causes the I/O control system to assert **DispStb/.** The contents of the I/O register select a particular display element and also determine what is displayed in that element.

In the case of the HP9810, an element is one of the 45 digits., and the segments are individually controlled by bits in the I/O register. In the HP9820 it's one of the 80 columns that make up the 16 characters, the LEDs in that column are again individually controllable. The HP9830 display has a hardware character generator, the contents of the I/O register select a pair of characters 16 positions apart in the display and determine the characters to be displayed in those 2 positions.

The 3 display systems will now be described individually.

4.1 HP9810 display

There are in fact 2 versions of the HP9810 **Display PCB**. The earlier one is fitted with 45 separate 7-segment display devices, the later one has nine 5-digit display modules, similar to those used in the 'classic series' handheld calculators. The PCBs are electrically very similar, the following description strictly applies to the later, more common, version, but it will be of use in repairing the earlier one too.

The bits of the I/O register are used in the following way :

DO(0)...DO(3) : Select a particular digit in a display row (binary en-

coded)

DO(4)...DO(6) : Select one of the 3 rows (1-of-n encoded)

DO(70), SO(0)...SO(3), CO(0)...CO(2) : These 8 bits give the segment pattern for the selected digit.

The display PCB has a separate anode driver circuit for each display row and a common cathode driver circuit for all 3 rows.

There are 24 essentially identical anode driver circuits, I shall describe the one for the 'a' segment of the X register display, the others work in the same way.

The **CO(1)** bit from the I/O register is buffered by U7c, giving **Segment(a)**. **DO(4)**, which is low to select the X register display row, is inverted by U15a, giving the **XSel** signal. Therefore, the output of U11b is low when the 'a' segment of the X register is to be turned on. This then saturates the associated PNP driver transistor, applying 5V to the appropriate display anode pin through a 25.5Ω resistor.

The cathode driver is based around a pair of decoder ICs U12 and U13. **DO(0)...DO(3)** are buffered, e.g. by U4a, and fed to the inputs of these decoders. U12 is enabled by U10a when **Digit(3)** is low, while U13 is enabled by U10b and U13f when **Digit(3)** is high. The selected output of the selected decoder goes low, and pulls the appropriate cathode pin of the display low via an PNP emitter follower such as U6d.

To enable either decoder (and thus any of the cathode lines), **DispEn/** must be low. This comes from the display protection circuit based around the monostable U4. For **DispEn/** to be low, **DispStb** from the **I/O Interface** PCB must be low, but U4 must not have timed out.**DispStb** is set low by the **I/O Interface** to display a particular digit but if, due to some fault, it is stuck low, U4 will time out, disable the cathode drivers and protect the displays.

4.2 HP9820 display

The HP9820 display appears to the machine as an 80-column, 7-row matrix of LEDs. The bits of the I/O register control it in the following way :

DO(0)...DO(6) : The pattern of LEDs in the selected column

DO7,SO(0)...SO(3), CO(0), CO(1) : Select a particular column. This is not a simple binary code, instead **SO(3)**, **CO(0)** and **CO(1)** select a group of 10 columns and **DO(7)** and **SO(0)...SO(2)** select a particular column within that group.

An 80-column, 7-row LED matrix would appear to need 80 cathode (column) drivers and 7 anode (row) drivers, for a total of 87 circuits. However, electrically, the display is wired as 20-column 28-row matrix requiring 20 cathode drivers and 28 anode drivers. The anode drivers are further simplified by considering them as 4 groups of 7.

A particular group of anode drivers is selected by the decoder U6. For any of the outputs 0...3 to be asserted, both the C input (**DispStb/**) and the D input (output of the monostable U5) must be low. This is essentially the same display protection circuit as I described for the HP9810 display. U6, therefore, decodes the **CO(0)** and **CO(1)** signals, enabling a particular **RowSel** signal via one of 4 PNP transistors.

Each **RowSel** signal applies 5V power to the emitters of a further 7 PNP transistors. These are individually controlled by the low 7 bits of the I/O register via inverters such as U13f. When a particular bit of the I/O register is set, the output of the inverter goes low, saturating the transistor. The collectors of these 28 PNP transistors feed the display matrix anode connections via 16.5Ω resistors.

The cathode driver circuit consists of a pair of 1-of-10 decoder ICs, U11 and U8. When **SO(3)** is low, the outputs U7c and U7d are forced high, so none of U8's outputs are selected. U7a and U7b are enabled, so 4 bits of the I/O register are passed to U11. Similarly when **SO(3)** is high, the outputs of U7a and U7b are force high, disabling U11, while U8 decodes the same 4 bits from the I/O register. The 20 outputs of these decoders are buffered by PNP emitter followers such as U1d, and connected to the cathode lines of the display matrix.

4.3 HP9830 display

The HP9830 display system is considerably more complicated because the character patters are generated in hardware, rather than by the machine's firmware. The 16 bits of the I/O register are used in the following way :

DO(0)...DO(3) : Select a pair of display locations, 16 characters apart, that is 0 and 16, 1 and 17...15 and 31.

DO(4)...DO(7), SO(0), SO(1) : The character code to be displayed in the second selected location.

SO(2), SO(3), CO(0)...CO(3) : The character code to be displayed in the first selected location.

The display system consists of 2 PCBs. The **Display** PCB contains the LED displays themselves and the anode/cathode drivers. The **Display Driver** PCB contains the character generator circuits. I shall begin with the latter.

To start a display scan, **DispStb/** pulses low. This sets monostable U5a, and since **Char(0)** is low, U5b is triggers, asserting **DispEn/** (**Char(0)** is simply DO(0), buffered by U18b on the **Display** PCB). **DispStb/** also clears the scan counter U12, which is used to select individual columns in the a particular display character. This is decoded by U7, the outputs of which select a column of the character generator ROM U1.

The asymmetric clock oscillator built from the monostables U6a and U6b now runs, clocking U12. Thus the counter scans through the columns of the character. The AND-OR-invert data selectors such as U14b are controlled by the clock signal. When **Clk** is high, the character code from the first selected position is gated to the inputs of the character generator U1. The outputs of this ROM are fed into the latch U9 and become the **LRow** signals to the the **Display** PCB. When **Clk** goes low. that pattern is latched in U9, and the character code for the second position is applied to character generator inputs. The AND gates such as U11c are enabled, so the character pattern appears on the **Row** lines Due to the asymmetry of the clock, the actual output consists of a short period while U9 is updated, then a much longer period where the **Row** and **LRow** signals are stable, giving the column patterns for the current column of the 2 characters. The column counter U12 is then incremented, there is the short update period again, followed by a stable period when the next column of the 2 characters is output. And the end of the character scan, output 6 of U7 goes low, disabling the clock generator until U12 is reset by the next **DispStb** signal.

A similar trick to the HP9820 display is used to reduced the number of driver circuits required. Rather than a 7-row, 160-column matrix, the display is wired as a 28-row, 40 column one. The inputs to the display driver circuitry on the **Display** PCB therefore consist of :

3 bits, DO(0)...DO(2) to select 1 of 8 sets of 4 characters

DO(3) to select a particular pair of character in that set, and to gate the Row and LRow lines to the appropriate anode drivers

3 bits, Col(0)...Col(2) to select a particular column within the characters.

2 sets of 7 signals (Row and LRow) giving the patterns in the 2 selected columns

DispEn/ : An overall enable signal.

The **DO(3)** bit is buffered by U18d and the inverted/buffered by the 4 gates in U22 providing the anode driver enable signals**.** The **LeftEn** signals are high when the characters to be displayed are in the first 8 positions of each half of the display, the **RightEn** signals are high when they are in the second 8 positions. These signals enable NAND gates in the row driver circuitry. For example, U19d gates **LRow(0)** to the base of U1a when **LeftEnB** is high, thus controlling the anode line **DispRowA(1)**.

The low 3 bits of the I/O register are buffered (for example by U18b) and become the **Char** lines. The **Col** lines are decoded by U35, which mimics the operation of U7 on the **Display Driver** PCB. Together these signals control 5 further decoders, each selecting 1 of 8 display columns. U36, for example, is enabled whenever the leftmost column of a character is being displayed, the **Char** signals determine which display location is used. The outputs of these decoders are buffered and connected to the display cathode lines. The outputs of U36 enable the leftmost columns of the display characters, those of U37 the second-from-left and so on up to U40 which selects the rightmost columns of the characters**.**

Chapter 5

Keyboard Theory of Operation

The HP9800 keyboards are controlled by circuitry on the **Keyboard Encoder** PCB mounted on the underside of the keyboard itself. The HP9810 and HP9820 keyboards do not use conventional mechanical keyswitches; they use a rather unusual device based on PCB tracks forming a pair of pulse transformers, which will be described shortly. This device was not really suitable for machines where 2 keys may be pressed together so the HP930, which has a shift key, uses normal keyswitches.

Since all 3 keyboards are different, I shall cover them in separate sections, but first...

5.1 HP9810/HP9820 'keyswitch'

Under each key of the HP9810 or HP9820 keyboard is a pulse transformer formed from spiral copper tracks on the PCB. The transformer has 2 windings, each consisting of a pair of spirals, one on each side of the PCB, connected by vias. When a current pulse is sent through one winding (the primary), a voltage is induced in the other winding (the secondary).

On the bottom of the plastic key plunger there is a metal disk. When the key is pressed, this disk is brought close to the transformer windings on the PCB. It absorbs energy from the primary winding and therefore reduces the induced voltage in the secondary windings.

The transformers are connected in pairs by the PCB tracks. The primary - windings of each pair are connected in series, as are the secondaries, but in one pair, one of the windings is reversed. Therefore, when a current pulse is sent through the primary pair, the induced voltages in the secondary pair oppose each other and the resultant voltage is close to zero. When one of the keys is pressed, the transformer under that key produces almost no

output, so the resulting voltage from the pair consists of that produced by the other key's transformer. The polarity of the output voltage pulse depends on which key of the pair is pressed. Of course if both keys in the pair are pressed simultaneously, there is, again no output voltage, but as I mentioned above, this design of keyboard is not suitable for applications which require several keys to be pressed at the same time.

The primary pairs are arranged in an electrical matrix with a diode in series with each pair to prevent sneak paths. All the secondary winding pairs are connected in series to form the 'Sense Loop'. The **Keyboard Encoder** PCB mounted under the keyboard itself scans the matrix by sending a current pulse through each pair of primary windings in turn, and looks for an induced voltage in the sense loop. Which key is pressed is determined by which matrix location causes a voltage to be induced, and the polarity of this voltage.

5.2 HP9810 keyboard

The keyboard is scanned by circuitry on the **Keyboard Encoder** PCB. A master clock signal is provided by the free-running oscillator U6a and U6f. When no key is pressed, **KeyIn/** is high, so U4c is enabled and the clock is fed to the scan counter. This is a 7 bit ripple counter made from the D-type flip-flops in U10, U11, U13 and U14a.

The master clock signal is looped through 2 pins on the cable edge connector and applied (as **ClkIn**) to the monostable U8a. The output of this turns on the transistor Q5, providing a current pulse to drive the matrix of transformers. This current is fed through a loop of wire so that it can be monitored by a clip-on current probe – but since I do not own such an instrument, I can't say what the pulse should look like.

Bits 1 ad 2 of the counter are decoded by the NAND gates in U1. Each gate has an associated PNP row driver transistor Q1...Q4, which feeds the current pulse from Q5 into one of the matrix rows. The keyboard column lines are connected to the decoder U9, which grounds one of them based on the top 4 bits of the scan counter. Thus the keyboard matrix locations are selected in turn, and 2 current pulses sent through each pair of transformer primaries.

The sense loop is transformer-coupled to the the comparator device U12. Due to the fact that the ends of the secondary winding of the coupling transformer are in antiphase, the 2 sections of U12 respond to opposite-polarity pulses in the sense loop. These sections are enabled by the LSB of the scan counter and its inverse. When the output of U12 goes high, the scan counter contents identifies the key uniquely, the LSB distinguishing between the 2

keys at the same matrix location.

The output of U12 triggers the monostable U8b (enabled by **RowDrv/**) causing **Key** to go high and **Key/** to go low. The latter reduces the threshold on the comparator U12, providing hysteresis. **Key/** is looped to **KeyIn** on the cable connector so that when a key is pressed, U4c is inhibited, halting the scan counter at that key's location. Thus the monostable U8b is continually retriggered, since the pressed key location is fed with current pulses on each keyboard master clock cycle.

KeyIn, which is simply the **Key** output from U8b after being looped through the cable connector asserts the **KeyDn/** interrupt line via U16a. It also enables the output gates (e.g. U7a) so the scan counter contents can be read into the processor I/O register. When the key is released, U8b times out, removing the interrupt signal and allowing the keyboard scan to resume. NAND gate U15 detects when the 'Stop' key is pressed and provides a reset signal to peripheral devices.

The output of the HP9810 keyboard is, of course, the 7 bit contents of the scan counter. The keyboard matrix is wired so that all the 'programmable' functions produce keycodes with the MSB clear, whereas keycodes with the MSB set correspond to system control keys that cannot be stored in a user program. Thus 6-bit wide RAM is sufficient for storing user programs.

The LEDs on the HP9810 keyboard are very simple. The top 8 bits of the processor I/O register are latched in U2 and U3 by **LEDStb** from the **I/O Interface** PCB. The LEDs are connected (with suitable series resistors) between the outputs of these latches and the +5V rail. Six of the 10 LEDs are independently controlled, the remaing 4 form 2 pairs (Run and Program; Float and Fix), each controlled by 1 bit of the latch. In either such pair one or other of the LEDs is always on.

5.3 HP9820 keyboard

The HP9820 **Keyboard Encoder** PCB is similar to that in the HP9810 machine. A master clock is formed by U8a and U8f. The output of this, **ClkOut** is looped through the cable connector whereupon it becomes **ClkIn**, and when no key is pressed, it clocks the 7 bit scan counter via U5c. The scan counter is, as before, a ripple counter made from the 7 D-type flip-flops in U11, U12, U7 and U15a.

ClkIn also triggers the monostable U10a, providing **RowDrv** to a PNP transistor. As in the HP9810, bits 1 nd 2 of the scan counter are decoded by the 4 NAND gates in U4, each of which drives one of the keyboard row driver transistors, feeding the current pulse onto a keyboard row line. Since

the HP9820 keyboard has more column lines than that of the HP9810, the column selection decoder requires 2 devices, U2 and U3.

The sense comparator circuitry based round U14 again follows the same pattern as that in the HP9810. When the pressed key is detected, the output of U14 triggers the monostable U10b, which as before reduces the threshold on U14. The outputs of U10b are looped through the cable connector to become **KeyIn** and **KeyIn/.** The latter halts the scan counter by inhibiting U5c. **KeyIn** asserts the **KeyDn** interrupt via U13d. The processor can set U15b using **LEDStb** to remove this interrupt signal and allow other interrupt to be detected, U15 is automatically cleared again when the key is released, ready for the next keypress.

U5b enables the data output buffers when a key is pressed and when the **CO** peripheral address outputs from the I/O register contain 1100_2. The outputs buffers, such as U9a, feed the scan counter contents to the I/O register data inputs. As in th case of the HP9180, the NAND gate U16 detects the 'Stop' key being pressed and asserts the peripheral reset signal.

5.4 HP9830 keyboard

The HP9830 uses a conventional matrix of mechanical keyswitches, so the **Keyboard Encoder** PCB is somewhat different to those in the other machines.

A free-running master clock is formed by U5f and U5a and buffered by U5c. The clock is divided by 2 by the first section of counter U2, and thus the outputs of U6c and U6d form a non-overlapping 2-phase clock. When no key is pressed, U10a is clear, so the clock at the output of U6d is gated to the 7-bit scan counter formed from U1 and the rest of U2..

The lower 4 bits of this counter are decoded by U4 and U17, which therefore ground each of the **KC** keyboard column lines in turn. The higher 3 bits of the scan counter select one of the **KR** row lines using multiplexer U3. Pressing a key connects a row line to a column line; when the location of the pressed key is addressed by the scan counter, the selected output of the column decoder grounds the active input of U3 through the closed keyswitch, and so the **Key** signal goes high and **Key/** goes low. **Key/** triggers monostable U10a, enabled by the clock at the output of U6c, stopping the scan counter**. Key** triggers the debouncing monostable network U10b and U11 and the output of U13c goes high.

This asserts the **KeyDn** interrupt signal via U9c. U14b can be set by the processor using **LEDStb** to disable the U9c and allow other interrupts to be checked for, in the same was as in the HP9820.

To read the keycode, the **CO** peripheral select outputs of the I/O register are set to 1100_2. This causes the output of U15a to go high. Thus (since the output of U13c is high), the output buffers such as U8b are enabled by U15b, feeding the scan counter to the inputs of the I/O register. The shift key grounds the input of U16f, thus causing its output to go high, and this is transfered to bit 7 of the I/O register via U7a.

The **STP** peripheral reset signal is asserted by U9d when the key is pressed in row 7 (U12b output is high) and row 1 (U16e output is high). This is, of course. the 'Stop' key.

The 'Rewind' key is directly connected to the rewind flip-flop on the **Tape Controller** PCB, it is not processed by the **Keyboard Encoder**.

The power-on indicator on the front of the HP9830 is simply a filament lamp powered from the 5V rail with a 22Ω series resistor. Anybody who understands the rest of the machine will have no problems with this.

Chapter 6

Printer theory of Operation

All HP9800 calculators have a hard copy facility. The HP9810 and HP9820 have an optional internal thermal printer. The HP9830 has a built-in interface for an HP9866 printer which may be stacked on top of the machine. The HP9866 itself is outside our scope, but the internal printer and the interface will be described.

6.1 HP9810/HP9820 thermal printer

The internal printer of this machines consists of a mechanical system to feed the thermal paper past the printhead elements and 2 PCBs of electronics. The **Printer Interface** PCB is mounted on the right hand side of the mechanism. The **Printer Driver** PCB is located under the mechanism and is plugged into an edge connector on the **Printer Interface** PCB. The printhead is connected to a flexible printed circuit ('flexiprint') that is an integral part of the **Printer Driver** PCB, while the motor and switches connect to an edge connector at the rear of this PCB.

The printhead has 80 elements and prints a horizontal row of dots for up to 16 characters across the paper. The paper is then advanced by one dot line and the next row of dots for the line of characters is printed. This is repeated until the whole characters have been printed

6.1.1 Printer Mechanism

The printer is mechanically quite simple. The platen is a rubber roller at the front of the chassis, with a second shaft beneath it carrying 2 belts that run along the bottom of the paper well and round idler rollers towards the rear. These 2 shafts are geared together by the gears on the right hand side

of the machine. When the platen revolves, it feeds the paper through the printer. A new roll of paper is loaded by the belts which feed the free end of the paper towards the platen.

The mechanism is driven by a stepper motor at rear of the unit. This drives a pinion on the lower front shaft via a toothed drive belt. This pinion is free to rotate on the shaft, but it meshes with a gear on the platen shaft. Thus the motor drives the platen and advances the paper.

The printhead is pressed against the platen by spring, thus keeping the resistive heater elements in contact with the thermal paper. When the 'Feed' key is pressed, the printhead is lifted off the paper by a linkage in the printer to reduce wear and to ease paper loading. The printhead operates a microswitch at the front of the unit when this occurs, this microswitch causes the electronic section to energise the motor.

The paper-out detector is simply a set of contacts that are held apart by the paper. When the paper runs out, a circuit is completed through these contacts.

6.1.2 Printer electronics

The electronic part of the printer consists of 2 main parts – to drive the printhead elements and to drive the motor. There is also some control circuitry common to both sections.

We shall begin with the printhead driver. The I/O register in the processor is extended by a further 10 bit shift register consisting of U4 and U2 on the **Printer Interface** PCB, which are serially loaded with the **I/O In** signal (which is, of course, the LSB of the I/O register inverted by U20d on the **I/O Interface** PCB). This forms a 26 bit shift register which is loaded by the processor using 2 I/O instructions.

The 80 printhead elements are electrically connected as a 20*4 matrix, with diodes on the **Printer Driver** PCB to prevent sneak paths. The 4 rows of this matrix correspond to 4 quarters of line of dots that the printer is currently printing, the 20 columns corresponds to the 20 dots within that quarter. The 20 column lines are driven from 20 bits of the shift register. The top 10 bits of the I/O register in the processor are inverted and buffered, e.g. by U8a and Q11. The 10 bits from the extra shift register are buffered, for example by U3a and Q10 (U3a is not an inverter, since the serial input to this extra shift register is inverted with respect to the I/O register). Thus the 20 **Col** signals can individually controlled.

The row selection circuit is on the **Printer Driver** PCB. Decoder U4 decodes 2 of the bits of the processor I/O register and enables one of the driver circuits such as Q15 and Q1. This grounds one of the **HDCom** signals. The

decoder is enabled by a pair of signals **PrtHdEn/** and **PrtPulse/** from the **Printer Interface** PCB. Both must be low to enable any of the elements in the printhead. When this occurs, one row of the electrical matrix can be selected and the elements in that row turned on by the bits in the shift register.

PrtHdEn/ is asserted by U7a on the **Printer Interface** PCB when the **PrtEn** signal is asserted by the **I/O interface** PCB and the feed button is not pressed. If the paper has not run out, **PrtClkEn/** will be low, enabling monostable U10. This monostable is triggered by **PrtHdEn/**, inverted by U7c. U10 provides the **PrtPulse/** enable signal.

The motor is advanced one step by the **MotorClk** signal (we will see how this actually drives the motor in a moment), which can be produced either by the processor during printing, or by the printer electronics for manual feed operations. During normal printing, **Feed** is deasserted. The output f U9c on the **Printer Interface** PCB thus goes low when the output of U9a is high, that is when the last quarter of the line is selected. When this occurs, the motor is advanced by the trailing edge of **PrtPulse/**, gated via U9b.

For manual feeding. the microswitch in the printer mechanism applies 24V to **FeedSw**. Thus turns on Q24, asserting **Feed**. U7a is inhibited, so the printhead cannot be driven. U9d's output is forced low, enabling U10 (even if the paper has run out, so that a new roll of paper can be loaded). U7d is enabled, gating the output of the astable multivibrator Q21, Q22 into U7c and thus repeatedly triggering U10. Thus the motor circuit is repeatedly clocked and the paper is advanced.

The same signals that trigger U10 also trigger the monostable U5b which has a much longer time constant. This provided an enable signal, **MotorEn/**, to the motor drivers and reduces power dissipation by turning off the motor windings when the printer is not being used.

The motor driver circuit is on the **Printer Driver** PCB. It consists of the 2 JK flip-flops U3a and U3b. The outputs of U3a and U3b which are in phase quadrature, are buffered and connected to the stepper motor windings. If **MotorEn/** is high then the 'lower' transistor of each buffer is certain to be cut off (for example U2c will ground the base of the lower transistor of the **MA** buffer under these conditions), thus ensuring that one end of each motor winding is floating. Therefore, as mentioned earlier, the motor windings then carry no current.

The **PrtFlag** signal to the **I/O Interface** PCB is produced by U7b on the **Printer Interface** PCB. It acts as a 'ready' signal indicating the printer is ready of the next operation. Normally U5a has timed out, so **PrtFlag** is produced by the trailing edge of **PrtPulse/**. But when the last part of the pine is printed and the motor steps the paper on one line, U5a is triggered.

This delays the **PrtFlag** signal until the trailing edge of U5a's output, when it times out. This gives the mechanism time to move the paper before the next line of dots is printed.

When the paper runs out, the contacts in the mechanism apply 24V to the **PaperOut** signal. This turns Q23 on, forcing the output of U9d high if the manual feed key is not pressed. The various monostables are now inhibited, which disables the printhead elements, prevents the motor from turning, and prevents **PrtFlag** from occurring. The last is used to indicate to the processor that there is no paper in the printer. As mentioned earlier, the manual feed switch will enable U9d even when **PaperOut** is asserted, enabling the motor to be used to load the new roll of paper.

6.2 HP9830 printer interface

The printer interface located on the HP9830 **I/O Backplane** PCB is similar in operation to an external interface plugged into one of the I/O connectors, and is quite simple

Select code 15 is decoded from the **CO** outputs by U1a. Its output goes low when the printer interface is selected. When **CEO/** is also asserted, the output of U2b goes high, clocking 7 bits of data from the I/O register into U4 and U5. The outputs of these latches are fed to the printer connector.

The output of U2b is inverted by U2d and sets the flip-flop U3a, thus asserting **PrtStb** via U2c. **PrtFlg** from the printer clears U3a when the printer is ready for further data.

The **SI(O)** input to the I/O register is pulled low by U1b when the device is selected (that is U2a's output is high because that of U1a is low) and the printer is ready for more data (U3a is cleared).

The printer signals are routed through an edge connector on the top of the I/O backplane to the 19 pin MIL-C-26482 connector on the rear of the machine.

Chapter 7

Storage System Theory of Operation

Since the HP9800 machines use volatile semiconductor RAM, a magnetic backing storage system was fitted to all models so that programs and data could be saved and reloaded. The HP9810 and HP9820 have a magnetic card reader, the HP9830 has a digital cassette tape drive.

7.1 HP9810/HP9820 Magnetic card reader

The magnetic card reader records 3 data tracks and a clock (also known as 'timing' or 'strobe') track in each direction on a magnetic card. The system consists of a motorised mechanism to pull the card past the 4-coil head and the **Card Reader Interface** PCB which links the mechanism to the processor **I/O Interface** PCB.

7.1.1 Card reader mechanism

The card reader mechanism consists of a 24V DC permanent-magnet motor with an integral gearbox, the output shaft of which is coupled to the card feed roller via a belt drive. This roller draws the magnetic card past a read/write head. Four optical sensors, each consisting of a filament lamp and a photodetector located so that the card blocks the light beam as it passes detect the leading and trailing edges of the card and also the presence or absence of a notch in the leading edge of the card, used to write-protect it.

7.1.2 Card reader interface

The **Card Reader Interface** PC is quite simple because most of the timing is controlled by the processor firmware..

There are 2 outputs from the **I/O interface PCB** used to select the card reader. **CROut** is asserted by U19a on this PCB when **I/OCmd(3)** and **Q(1)** are both high, that is when an 'STC' machine is executed with internal select code 2. **CRSOut** is asserted by U1c when **Q(1)** and **RdInst** are high. The latter signal is asserted by U8d on the **Clock** PCB when **Q(9)** is low and **I/OCmd(0)** is asserted. So **CRSOut** is asserted when an 'STF' instruction with select code 2 is executed. These 2 signals write the low 6 bits of the processor I/O register into a 6 bit output port built form D-type flip-flops on the **Card Reader Interface** PCB.

Bit 5 of the I/O register is loaded into U5a by **CROut**. The output of this flip-flop is buffered by a 2-transistor circuit which connects the positive side of the motor to the +12V supply when the flip-flop is set. Since the negative side of the motor is connected to the -12V rail, 24V appears across the motor which thus rotates.

The card sensor photodetector signals are amplified by 2-transistor circuits and fed to **DI(3)...DI(6)**. These transistors are cut off by **SenseEn** when the motor is not running. 2 of the sensor outputs are diode-ANDed to give the **RdEn** signal which enables the read amplifiers.

Bit 4 of the I/O register is latched in U8a, again by **CROut**. The output of this flip-flop, **WrD**, disables the write drivers when it is set. It appears to be set throughout read operations but cleared during write operations.

The 4 write circuits are essentially the same, so only track A will be described. Bit 0 of the I/O register is copied into U8b, which drives the inverter U7c. When U8b is set, the output of U7c is low, saturating the transistor and making the voltage on its collector close to +12V. The coupling diode is reverse-biased, so no current flows in the head winding. When U8b is set, the transistor is cut off, and a current flows through the head from the =12V supply rail via the coupling diode and 2.7k resistor. U9e clamps the head drive signal to ground, preventing writing during read operations.

There is one difference for the 'S' clock write channel. The output flip-flop, U5b, is loaded by **CRSOut** rather than **CROut**. This makes it easier to delay the clock pulse with respect to the data pulses.

The 4 read amplifiers are, again, identical. The signal from the head is amplified by an operational amplifier such as U4, and fed to a discrete-transistor amplifier and schmitt trigger circuit. The latter is enabled by **RdEn** from the card sensor circuitry when there is a card in the reader track.

The output of the clock channel is fed to a 2-transistor circuit which produces a positive-going pulse on **CRRdClk** for a magnetic transition of either polarity on the card. This signal loads the outputs of the 3 data channels, suitably buffered, into the read flip-flops such as U10b. These can be read into the processor I/O register via the associated open-collector inverters, such as U9c. When there is no card in the reader, **RdEn** being deasserted asserts **ClrRd**, forcing all the read flip-flops to be set and thus the outputs of the inverters to be in the high-impedance state. Thus other devices can drive the i/O register inputs.

CRRdClk/ is fed to the **I/O Interface** PCB where it sets the SR flip-flop U4c, U4d. This causes the first gate of U9 to assert **I/O flag** when a card reader I/O instruction is executed. This is used to indicate the presence of new data in the read flip-flops. This flip-flop is cleared by U2d when **I/OStb** is asserted at the end of an I/O instruction with bit 9 set (this is detected by U2a on the **Clock** PCB along with **Q(1)** being set, that is select code 2.

7.2 Cassette drive

The digital cassette drive fitted to the HP9830 uses digital-grade tape in cassettes similar to audio Compact Cassette. Unlike normal audio cassette recorders, though, the tape is transported simply by turning the spools, there is no capstan and pinch roller. Although this means the tape does not travel at constant speed, the encoding system used on the HP9830 in not sensitive to this.

The entire width of the tape is used at once, the cassettes are not turned over like audio cassettes. Two tracks are recorded on the tape. A pulse on one track indicates a logic 0 bit. A pulse on the other track indicates a logic 1 bit. And a pulse on the 2 tracks simultaneously is used as a start-of-byte marker. This is known as 'Bit-Mark-Sequence' or 'BMS' in some documentation

An optical EOT sensor detects the clear leader tape in the cassette. It consists of a small filament lamp and light-dependent resistor (LDR) mounted so that light from the bulb is reflected off the cassette shell and onto the LDR when the clear leader (rather than opaque tape) passes in front of the sensor assembly.

7.2.1 Motor control

The tape drive is fitted with a pair of permanent-magnet DEC motors, one for each tape spool. The motors are mounted on a metal beam, pivoted at

the mid-point. This beam is rocked by a pair of solenoids, and this engages the motor spindle with a rubber-tyred wheel attached to each tape spindle.

The **Motor Control** PCB selects a particular motor and its associated solenoid to control the tape direction and regulates the speed of that motor. The direction control system takes in 2 signals from the **Tape Controller** PCB. **MotorOn/** goes low to start tape motion, causing the output of U4c to go high, enabling U4b and U4d. The **Dir** direction control signal, inverted by U4a thus causes the output of one of those gates to go low., turning on the associated PNP transistor. This drives a pair of NPN transistors, one of which energises the solenoid, the other grounds the negative side of the selected motor.

The motor speed controller senses the current through the selected motor by measuring the voltage drop across a 10Ω resistor in series with it (this is the resistor connected between TP2 and TP3 on the **Motor Control**) PCB. This voltage is amplified by U1 and compared with a reference level by U2. The output of U2 controls the motor pass transistor, Q9 on the **PSU** PCB (it is located there simply because it needs to be fitted on a heatsink), thus controlling the motor current and therefore the motor speed.

The **Speed** input from the **Tape Controller** PCB is applied to U3a, the output of which varies the reference voltage level to U2. When **Speed** is low. the reference voltage is increased, so the motor turns faster.

The output of the LDR in the optical EOT sensor is converted to a digital signal by U1 on the **Tape Drive Connector** PCB acting as a comparator. When the tape leader is detected, **EOT** goes low, causing the output of U1a on the **Tape Interface** PCB to go high. If the motor is running, monostable U6 is triggered, and when this times out, U5b is set (since the output of U4b must be 1 when **MotorOn/** is low). This asserts **SpeedClamp** via U4c.

When **EOT** goes high again (tape, rather than leader in front of the sensor), U5b is directly cleared, deasserting **SpeedClamp**. U5b is also set when there is no tape in the drive (**CassSw** is low) or during power-on initialisation (**ClrMark**, from U17f on the **Tape Controller** PCB is low).

Back on the **Motor Control** PCB, **SpeedClamp** causes U3b to clamp the motor speed control line to the voltage of its associated zener diode. This prevents the motor running too fast and causing damage when the tape stalls.

7.2.2 Low level read/write system

This section translates between the conventional clock and data signals and the BMS format described earlier. It also interface to the tape head itself.

For a write operation, the output of U5b on the **Tape Controller** PCB goes high, asserting **Write.** On the **Read/Write** PCB, U4c ANDs this with the output of the write-protect switch (**WpSw/**), preventing writing on protected cassettes. **WrEn** enables the write encoder circuit.

During writing, if **WrData** is high when **WrClk** goes high, the output of U8b goes low, and forces **WA** high via U7a. Similarly, if **WrData** is low (making the output of U6a high) when **WrClk** goes high, the output of U8c goes low and that of U7b, **WB** goes high. Finally, when **WrMark** is asserted, the output of U8a goes low, asserting both **WA** and **WB**. In this way the incoming data is converted to the BMS format.

The head drivers, located on the **Tape Head** PCB are identical in operation, so only the 'A' track will be described. **WA** is inverted by U1 and thus the outputs of U1b and U1c are in antiphase. This provides the signal to the head winding. The power to U1 is controlled by a transistor on the **Read/Write** PCB which is turned on via U6e during writing, there is no write current in the head at other times.

The 2 read amplifiers are identical, so as before I will only describe the 'A' channel. The output from the head winding is amplified by U2a on the **Tape Head** PCB and fed, as an analogue signal (**RA**) to the comparator U1b on the **Read/Write** PCB. The threshold of this comparator is determined by its current state via the 2 transistor circuit connected to its output, thus providing hysteresis and also by the state of the **Speed** signal via U6f, U4d and the associated transistors. When the tape is running fast for searching, the comparator threshold is increased, improving the noise immunity. The output of U1b is fed to U4a, which is enabled during reading by U6e. The **TTLRA** and **TTLRB** signals at the outputs of U4a and U4b are logic signals that indicate when pulses have occurred on the tape tracks.

When there is a pulse on both tracks at the same time, the output of U7c goes low, clearing U3a. This asserts **RdMark.**

When a pulse occurs on track 'A' and no mark has been detected, U7d clears U5b, thus making **RdData** high. When a pulse occurs on track 'B' or a mark has been detected, U5c sets U5b, making **RdData** high.

A pulse on either (or both) tracks makes the output of U5a pulse low. This triggers the monostable U2. During reading, when no mark has been detected, the output of U5d is high, and thus the output of U6b low. When U2 is triggered, therefor, its Q/ goes low causing **RdClk** to pulse low via U5b and U6c.

Thus the BMS data on the tape is separated into clock, data and mark signals.

7.2.3 Tape control

The **CO(n)** peripheral select lines are decoded on th **Tape Interface** PCB, and when select code 1010_2 is detected, the output of U13a goes high. If **CEO** is also high, U11c asserts **StatEn** enabling the gates in U7, which pass various status signals to the **SI(n)** inputs to the I/O register. If **CEO** is low, **TapeSel** is asserted by U11b, and, this, after being buffered by U11a, enables the data output gates in U8 and U9 so the tape data can be read into the I/O register (the source of this data will be explained shortly).

On the **Tape Controller** PCB, **TapeSel** triggers monostable U10. U10 enables decoder U8, which decodes **SO(0)...SO(2)** and asserts **RunCmd/** for any of the 4 read commands or the write command. This latches the **SO(n)** lines in the command latch U1. The 2 least-significant bits of U1are fed to the **Motor Control** PCB via U6a and U6b, to control the tape speed and direction.

RunCmd also sets the run flip-flop U4b, which asserts **MotorOn** via U6c, starting the tape motion. The run flip-flop is cleared at the end of the tape (**SpeedClamp** goes high, thus clocking a 0 into U4b via U16a and U17e), or when **Stop** is asserted by U12c. This occurs at power-on (from U17f), when a stop command is given to the interface (**StopCmd/** is set low by U8), or when there is no tape in the drive.

Manual rewinding is controlled by U4a. This is set by the 'Rewind' key on the keyboard, and its output causes the output of U5c to go high if there is no tape operation in progress. This forces all 3 motor control signals low, causing the tape to rewind at high speed. U4a is cleared either when a tape command is started (**RunCmd** clocks a 0 into U4a), or when the tape ends or when one of the 'stop' conditions mentioned earlier occurs.

7.2.4 Tape data register

Data is stored on the tape in 9-bit characters, each character is preceded by a mark. Of the 9 bits, 8 of them are transfered to or from the least significant byte of the processor I/O register. The centre bit is, however, loaded from bit 3 of the command latch U1 on the **Tape Controller** PCB and indicates the file marker character. The centre bit of the nine is used for this since it is in the same position relative to the marks no matter which way the tape is being read, enabling file markers to be found during both forward and reverse scans of the tape.

The data register consists of the D-type flip-flop U20b and the two 4-bit registers U19 and U18, cascaded in the usual manner. U15a and U15b provide a parallel load facility for the most significant bit.

The timing of the data transfer is controlled by the 10-state counter U3. U7c asserts **WrMarkEn/** when this counter gets to its final state of 1001_2. Data bits are transfered in the 9 states when **WrMarkEn/** is not asserted.

Although these 2 sections are common to both reading and writing data, it is simpler to consider their operation and that of the associated control in these 2 cases separately

Tape writing

When the tape system is selected by the CPU during a write command, **LdSR** is asserted by U5a, thus transferring data from the processor I/O register to the tape data register.

Since **write** is asserted by U5b during write commands, the write clock oscillator U2 and U9 is enabled, The reset-to-9 input of U3 is enabled (and the reset-to-0 disabled via U17d) so when the write operation is started, U11b is set by the rising edge of **write**, forcing U3 to state 1001_2 and asserting **WrMarkEn/**. This enables U13b via U14f, casing the first **RawWrClk** pulse to assert **WrMark** and thus write a mark on the tape. **WrMark** also clears U11b via U14d, thus enabling the counter U3 to count.

While **WrMarkEn/** is not asserted, U7d is enabled, and the shift register is clocked by the write clock oscillator. U7d also provides the **WrClk** signal to the data encoder described above. The data input to the encoder is connected to the least significant output bit of the tape data register, Thus the 9 bits in the shift register are transfered to the tame, then **WrMarkEn/** is asserted to write the marker for the start of the next 9 bit word, and the sequence repeats.

Tape reading

When the tape is read, the write clock oscillator is disabled since **Write** is low. The output of U7d is forced high, and thus U7b passes the read clock signal, **RdClk** from the data decoder to the shift register. When a data bit is detected on the tape, it is shifted into the data register, The bit counter U3 is also clocked by **RdClk**, and is set to 0 when U20a is set by a mark being read from the tape. **WrMarkEn** is asserted by U7c when 9 bits have been read from the tape.

The parallel outputs of the data register are transfered to the processor I/O register via the data output gates in U8 and U9 on the **Tape Interface** PCB, enabled by U11a

7.2.5 Tape flag and interrupt

The tape flag signal is produced by U12b on the **Tape Controller** PCB. When the tape system is writing the output of U6d is forced low, forcing the output of U12a high. Therefore the flag signal is controlled by U15c. Its output goes low, causing a flag, when **WrMark** is asserted, indicating that a new word should be loaded into the shift register. U11a is set at the start of a write operation, inhibiting the flag signal for the mark that is then generated.

During reading, U15c's output is always high. The flag is generated by U12a, normally when 9 bits have been read from the tape, since U7a's output is forced high by the **Bit4** output of the command latch being low. However, by setting this bit, the tape flag will only be asserted when **Bit4Out/** from the data registers is asserted after 9 bits have been read, that is when a file marker has been detected. As mentioned above, this is independent of the direction of tape movement.

The tape flag signal sets U5a on the **Tape Interface** PCB, thus asserting **BusI/ORd/**. This causes the output of U7c on the **I/O interface** PCB to go high, triggering the monostable U8 and copying the data from the tape data register into the I/O register. U8 also sets the flip-flop U13a, asserting the processor I/O flag via the second gate in U9.

During a scan for file markers (that is, **Bit4** is high and therefore so is **BuffBit4**), the tape system can interrupt the processor. When the tape system is not selected by the processor and **BuffBit4** is high, the reset input of U14a is deasserted. This flip-flop is therefore set by a rising edge at the output of U2c,which can occur in 3 ways : if the tape runs to the clear leader, and thus **BuffSC** is asserted causing the output of U1c to go low; if the cassette is removed, causing the output of U1b to go low; or if the output of U2b goes low, which occurs when the file marker character is found. The last occurs when bits 2...5 of the tape data register are all set and a tape flag occurs, but since **Bit4** must be high to enable the interrupt system in the first place, the centre bit (**Bit4Out/** of the shift register must also be asserted to enable **TapeFlag/** as described above.

Interrupts other than the file marker being found cause the output of U3a to go low, setting U5a and causing **BusI/ORd/** to be asserted, which has the effects described earlier.

When the interrupt occurs and U14a is set, **KeyDn** is asserted via U3c, interrupting the processor. U3b pulls **SI(1)** low, thus enabling the processor to determine the source of the interrupt.

The output of U14a, **StopTape/** causes the output of U12c on the **Tape Controller** PCB to go low, thus clearing the run flip-flop U4b and stopping

the tape.

7.2.6 Beeper

The HP9830 incorporates a fixed-frequency, fixed-duration beeper which is described here simply because it is located on the **Tape Interface** PCB.

The otherwise-redundant **CROut** signal (used for card reader output in the HP9810 and HP9820 and described in that section) triggers the monostable U12 when a beep is required. This enables U15c, passing the output of the oscillator U15a, U15b to the loudspeaker driver transistor via U15d. Thus the speaker is driven. The positive side of the speaker is connected to the output of an emitter follower, the base of which is connected to a 22μF capacitor. When U12 is not triggered this capacitor is charged via the diode and 470Ω resistor. When U12 is triggered and the beep is produced, this capacitor discharges via the 3.9kΩ resistor thus gradually reducing the voltage applied to the loudspeaker and giving a decaying amplitude envelope.

Chapter 8

Power Supply Theory of Operation

The power supply provides the following DC voltages from the AC mains input :

+5V : Logic power supply

+12V : Supply for the ROM selection inputs, and for the analogue parts of the storage system.

-12V : Used to provide the ROM bias supply and also for the analogue parts of the storage system

+16V : Supply for the PMOS RAMs

+20V or +19.5V : Actually referenced to the +16V supply, this is the bias rail for the PMOS RAMs.

+24V : This is only present in the HP9810 and HP9820 machines, and is used for the printer motor and thermal printhead.

The power supply also provides the **Init/** signal which sets the processor microcode program counter to 1717 when the power is first turned on.

The power supply consists of the mains input circuitry mounted mostly on the rear panel of the machine., This feeds a double-wound transformer which supplies lower voltage AC to the PSU PCB(s). The outputs of the transformer are rectified and smoothed, and then regulated down to the voltages given above. In all cases, the +5V regulator is s switch-mode design, while all other voltages are produced by linear regulators.

Although the HP9830 power supply is physically different from that in the other 2 machines, the circuitry is very similar and thus all the power

supplies will be described together. The main physical difference is that in the HP9810 and HP9820, the transformer secondary windings are wired to a pair of edge connectors mounted in the base of the machine. Two of the power supply PCBs plug both into those connectors and into connectors on the main backplane, These PCBs connect to the appropriate transformer secondary connections on the case-mounted connectors and provide the DC voltage to the main backplane. One of these PCBs also passes AC to the third PSU PCB which only plugs into the main backplane. In the HP9830, the transformer secondary windings are connected to a free edge connector which is plugged into the rear edge of the main backplane, The AC outputs of the transformer are fed over the backplane to the single PSU PCB, the regulated outputs from which are again fed over the main backplane.

8.1 Mains input and transformer

AC mains enters the machine via a standard IEC60320 C14 plug[1]. The live (phase) wire is fused at 6A, and the supply is connected to 2 or 3 (depending on the machine) IEC sockets (EN60320-2-2 Sheet F type) which can be used to power peripheral devices. Mains is also fed through an interference filter circuit (which may either be an encapsulated module or discrete components) and then to an in-line connector for the mains switch fitted to the keyboard.

When the switch is turned on, the mains is then applied to the primary of the power transformer via a further fuse (1A for 220V or 240V mains, 2A for 110V or 120V mains). When the switch it turned off, a 1MΩ resistor is connected across the output of the mains filter, ensuring the capacitors are discharged when the mains plug is removed and preventing (admittedly very minor) electric shocks from the exposed plug pins.

The mains transformer has 2 primary windings, each 120V with a tap at the 100V position, A pair of slide switches on the rear of the machine connects these windings for different mains voltages as follows :

100V : Mains is supplied to the two 100V sections of the primary windings connected in parallel

120V : The two 120V primary windings are connected in parallel across the mains

[1]This is a stanard mains input connector as used on many other devices, Miud you, if you have trouble finding where to feed the mains in I think you will have even greater trouble repairing the processor.

220V : One entire 120V winding and the 100V section of the other winding are connected in series. Mains is connected to the ends of this combination.

240V : Both primary windings are connected in series across the mains.

The machine's cooling fan has a 120V AC (shaded-pole) motor and is connected across one of the 120V primary windings. For 120V mains, it ends up being simply connected to the mains, for other mains voltages the transformer primary windings act as an autotransformer to supply 120V to the fan.

Whether the mains transformer has one multiply-tapped secondary winding, or several with the centre taps joined together is something that I have no determined. Electrically it makes no difference. The center-tap of the winding is connected to the system ground rail via the PSU PCB, symmetrical outputs voltages from the transformer are bi-phase rectified and smoothed by the PSU circuitry.

8.2 Regulators

The regulators are based round a type 723 IC, which contains a reference voltage source and an error amplifier. The operation of a linear regulator circuit is to compare a voltage derived from the appropriate output voltage with a known reference voltage using the error amplifier and use the output of the latter to control a pass transistor which in turn controls that voltage output of the supply. The +12V supply uses the internal reference of the associated 723, all other supplies use the +12V supply as their reference.

The DC inputs to the regulator circuits are produced by bi-phase rectifiers and smoothing capacitors.

All the voltage outputs apart from +19V have crowbar circuits. Each crowbar circuit consists of an SCR (thyristor) triggered by a zener diode. If the power supply voltage exceeds that of the zener diode, the SCR receives gate current, turns on, and thus shorts that power supply rail to ground, protecting the rest of the machine.

A couple of RPL programs

I investigated the operation of the regulators using my HP49G calculator, and found the following RPL programs to be helpful :

'PAR' calculates the equivalent resistance of 2 resistors in parallel :

```
PAR : \<< INV SWAP INV + INV \>>
```

'PDIV' calculates the voltage ratio for a potential divider :

```
PDIV : \<< SWAP OVER + / \>>
```

8.2.1 +12V regulator

Since the +12V supply is used as a reference for most of the other voltage rails, I shall begin with this one. It is based round U1 on the **+/-12V PSU** PCB in the HP9810 and HP9820, or U2 on the **PSU** PCB in the HP9830. The output of this regulator IC controls an emitter-follower pass transistor which in turn controls the voltage of the +12V output The **+12Sense** signal is connected to the +12V rail on the backplane, and measures the voltage there. This means that the voltage drop across the PCB edge connector is irrelevant, the voltage at the load is what is measured.

The operation of the 723 in this circuit is to try to make the voltages on its 2 inputs equal. The voltage on **In+** is easy to determine. It's simply the reference voltage, **Vref** from that IC, and the data sheet gives a typical value of 7.15V for this.

The voltage at **In-** is determined by the output voltage and a potential divider. The first thing to do is to calculate the division ratio :

```
499. 788. 11500. PAR PDIV
```

Now, due to the operation of the 722, the output voltage multiplied by that ratio will be the reference voltage. So :

```
7.15 SWAP /
```

will give the output voltage, which is very close to 12V. The 11.5kΩ resistor may have a different value in different machines to compensate for slight changes in the reference voltage of the 723.

The current sensing inputs of the 723, **IL** and **IS** are connected across the sense resistor (1Ω in the HP9810 and HP9820, 0.75Ω in the HP9830).

If too much current is drawn from the power supply, the increased voltage drop across this resistor will cause the 723 to remove the drive from the pass transistor, shutting the supply down.

8.2.2 -12V regulator

This regulator, being designed to give a negative output voltage, is possibly the most difficult to understand. The pass transistor is a PNP device and is biased on by the resistor connected between its collector and base. The output of the 723 (U2 on the **+/-12V PSU** PCB in the HP9810 and HP9820 or U4 on the **PSU** PCB in the HP9830) reduces the absolute vale of the the base voltage of this transistor, reducing the output voltage. As a result of this, the 'sense' voltage is applied to **In+** and the 'reference' voltage to **In-**. As before, **=12Sense** is connected to the -12V rail on the backplane.

The next thing to do is to calculate those two voltages. Let the output voltage be 'V' Then the voltage on **In+** can be calculated by :

```
4530. DUP PDIV 'V' *
```

Or you could notice that a potential divider with equal resistors divides the voltage in half.

The voltage on **In-** is first calculated relative to the +12V rail and then 12 is added to the result to make it relative to the ground rail :

```
3010. 9090. PDIV 'V' 12. - * 12. +
```

The operation of the 723 is to make those equal, so set them equal on the calculator and isolate 'V' :

```
= 'V' ISOL
```

This gives an output voltage close to -12V as expected.

In this negative regulator, the internal current limit system of the 723 cannot be used. Instead, the B-E junction of a PNP transistor is connected across the current sense resistor. When the voltage drop across this resistor exceeds 0.7V, the transistor is biased on, and removes the base drive from the pass transistor.

8.2.3 +16V regulator

The +16V regulator, based around U1 on the **+16V/+20V/+24V PSU** PCB in the HP9810 and HP9820 or U3 on the **PSU** PCB in the HP9830 is very similar to the +12V regulator. The operation of the 723, its pass transistor and the current limiter is identical.

In the case of the HP9810 and HP9820, the 'reference' voltage at **In+** is given by :

```
681. 953. PDIV 12 *
```

The operation of the 723 means that this is equal to the output voltage divided by the potential divider circuit connected to the **In-** input. This division ratio is calculated by

```
887. 698. PDIV
```

And of course

```
INV *
```

gives the output voltage, which is close to 16V

For the resistor values used in the HP9830, the output voltage can be calculated by :

```
499. 698. PDIV 12 * 887 698 PDIV INV *
```

which is again close to 16V

8.2.4 +20V (or +19.5V) regulator

The (very low current) memory bias supply is provided by a simple zener diode circuit 'on top of' the +16V rail. In the HP9810 or HP9820, the current is provided by a resistor from the +24V supply. In the HP9830, a constant current source based round Q10 on the **PSU** PCB is used for this purpose.

8.2.5 +24V regulator (HP9810 and HP9820 only)

This circuit is again similar to the +12V regulator. The **In+** input is held at +16V, and therefore the output voltage, determined by the potential divider on the **In-** input is given by :

```
16. 806. 1620. PDIV /
```

which is close to 24V. The current limiting circuit of this regulator senses the base current of the pass transistor, dropped across the 4.7Ω resistor, and is not particularly effective. A short-circuit from the +24V rail to ground often burns out the 0.51Ω series resistor

8.2.6 +5V regulator

The +5V supply uses a switch-mode regulator. The control transistor (known as the chopper) is either saturated or cut off and therefore dissipates little power. When the output voltage is too low (compared to the reference voltage), the 723 (U1 on the **+5V PSU** PCB in the HP9810 or HP9820 or on the **PSU** PCB in the HP9830) turns on the chopper, applying the raw input voltage to the load via the series inductors. Due to the inductance of the latter, the output voltage does not rise instantaneously, but relatively slowly as the magnetic field around the inductor builds up. When the output voltage exceeds the reference voltage, the 723 cuts off the chopper transistor. The magnetic field around the inductor begins to collapse, and supplies power to the load via the flyback diode. Thus the output voltage has a small ripple (essentially determined by the hysteresis of the 723 error amplifier) around the required voltage.

The **+5Sense** signal is connected to **In-** on the 723, and thus the output voltage is equal to the reference voltage, set by the potential divider on **In+** which is given by :

```
698. 511. PDIV 12. *
```

which is close to 5V. There is no overcurrent protection in the +5V regulator, but if the output is shorted to ground (possibly via the crowbar circuit) then the NPN transistor connected to the **Comp** input of the 723 will be turned on. This will clamp **Comp** to ground, disabling the drive to the chopper transistor and thus shutting the regulator down.

8.3 Init signal

The **Init/** initialisation signal used to set all bits of the microcode program counter is produced by a monostable circuit based round U2 on the **+5V PSU** PCB in the HP9810 and HP9280 or U5 on the **PSU** PCB in the HP9830. This is triggered by the resistor-capacitor network on its input shortly after power-on and during its time period it asserts **Init/** via the NPN transistor buffer connected to its output.

Chapter 9

Tools and Test Equipment

Obviously in order to repair the machine you need to dismantle it, but before I describe that procedure, a few words about the tools and test equipment that are necessary (or at least useful).

9.1 Tools

The basic tools needed to take an HP9800 apart and perform electronic repairs include:

Screwdrivers : Flat-blade, Phillips and Pozidriv screwdrivers are needed in various sizes. To prevent damage to the screw heads, make sure you know the difference between Phillips and Pozidriv, and use the correct one.

Spanners and Nutdrivers : The HP9800 machines are assembled with most American UNC fasteners. Therefore you need imperial-sized spanners to fit the nets. It is useful to have open-ended spanners, ring spanners and nutdrivers covering the range 3/16" to 3/8". A couple of such tools not found in most sets are an 11/32" open-ended spanner which is essential for removing the rear panel of the machine and a 1/8" nutdriver to remove the end cover from the card reader motor in the HP9810 and HP9820.A 9mm open-ended spanner can be used in place of the 11/32" one (11/32" = 8.73mm) if you can't get the latter, but the right tool is a lot better.

Hex keys : There are not many hex-headed screws in these machines, but there are some, and a set of imperial-size hex keys is necessary.

Pliers : Not to be used as a substitute for a spanner, these are needed for forming wire leads and similar applications

Small side cutters ; Most likely you will have to replace some components, and you will need to cut off the excess lead length on the new ones.

Small external circlip pliers : External circlips are used on the spindles in the mechanical sections of these machines, you will need to remove them.

Tweezers : Very useful for fitting small parts in the mechanical sections of the machine. Get non-magnetic stainless steel ones, there is nothing more annoying than a pair of tweezer that will not drop the part where you want it (Actually, there is, It's a pair of magnetic or magnetised tweezers that also manages to move other parts around for you).

Soldering iron and solder : The PCBs in these machines are generally double-sided (although a few are multiple-layer) with plated-through holes. I would seriously recommend a temperature-controlled soldering iron to avoid damaging the PCBs. Some thin (22swg or thinner) resin-cored solder is needed to fit the replacement components. Of course this is the 'real' lead/tin solder

Desoldering tool : Either a desoldering pump ('Solder Sucker') or braid, whichever you prefer. Remember that in most cases the PCB is much more valuable than the component that you are removing and it may be worth cutting the pins off the component and removing them one at a time from the PCB.

Drill : Either a hand drill of the wheelbrace type or a small electric drill, and a selection of twist drill bits. The connectors on the rear of the machine are riveted to an internal bracket, and if these need replacement you have to drill out the rivets.

UNC taps and dies ; Some of the screws were initially assembled with a locking adhesive, and in some cases, particularly in the keyboard assembly, they screw into tapped bushes in a brittle plastic moulding. I would recommend that you clear out the threads after removing them by running a tap into the hole and a die over the screw. Strictly these should be thread-clearing rather than thread-cutting tools, but in practice it makes little difference. You need (at least) sizes 4-40 and 6-32 UNC.

When we come to repairing the HP9810 and HP9820 card reader, and to a lesser extent the internal printer, I will, unfortunately, recommend the use of small machine tools.

It is well worth buying good quality tools. They will last a lot longer and more importantly will not 'chew' screw heads, damage PCBs, and the like. You are working on a valuable machine, it is worth taking care of it. I have often said 'I am not rich enough to buy cheap tools'.

9.2 Card reader roller puller

If you are repairing an HP9810 or HP9820 it is very likely you will have to replace the drive roller in the card feed mechanism, To remove the roller without damage you need a special puller tool, which unlike the tools mentioned above is not available commercially. It can be made in a reasonably well-equipped home workshop. Some means of doing coordinate drilling and light milling, for example a small lathe with a vertical slide, is very useful.

The puller consists basically of a steel 'cage' which fits around the old roller with a bolt to press on the end of the spindle and thus force the roller off.

The top plate of the tool is a piece of 1/4" thick steel measuring 2.5" by 2". Taking one corner as the 'origin' and the shorter edge as the 'Y axis', drill 4mm holes at (0.25, 0.25), (1.25, 0.25), (2.25,0.25), (0.25, 1.5) and (2.25,1.5) (all dimensions being in inches, of course). Then drill a 10.6mm hole (which is the tapping size for M12) at (1.25,1.5). and tap it M12. Although I have specified metric fasteners throughout this tool, it can obviously be made with any bolts of a similar size (for example1/2" UNC in place of M12, 6-32 or 8-32 UNC in place of M4) if the hole sizes are adjusted to suit.

The bottom plate is 1/8" steel of the same dimensions. Drill five 4mm holes at the same coordinates and a 1/8" (or 3.2mm) hole at (1.25,1.5) and then mill a slot from that hole to the nearest long edge. In use, the card reader spindle will be slipped into this slot with the bottom plate 'behind' the roller and an M12 bolt running in the tapped hole in the top plate will press on the end of the spindle

The plates are joined by 5 pillars made from 3/8" steel rod. Each pillar should be 2" long (the exact length is not too critical, but it is obviously important to make them all the same length). Each end of each pillar and drilled centrally at 3.5mm and tapped with an M4 thread (this is easy to do on a lathe with a drill bit held in a tailstock chuck).

The puller frame is assembled using M4 bolts in the obvious way. I found that hexagonal-socket ('Allen') cap-head screws were the easiest to use here.

It is worth making 2 puller bolts. One is simply an M12 bolt around 60mm long with the end faced off to make it perpendicular to the axis. The second is a bolt prepared in the same way and then a 3.3mm hole is drilled along the axis (from the end furthest from the head) to a depth of about 0.5", If you have a 'D-bit' to make a flat-bottomed hole then use it, otherwise a twist will is adequate.

In use, the latter bolt is used first with the reader spindle fitting into this hole. This will guide the spindle and hopefully prevent it getting bent. Once the roller has moved, if it still won't slide off by hand, the plain bolt can be used to move it further along the spindle.

9.3 Test equipment

Unfortunately you can't see the logic signals in the HP9800 machines directly, so you need a few items of electronic test equipment:

Multimeter : This needs to have DC voltage ranges covering the outputs of the power supply and AC voltage range to check the output of the power transformer. Resistance ranges to check for continuity and short circuits are essential too. Other ranges my be nice to have, but they are certainly not essential

Logic probe : Even with a logic analyser (see below) it is very useful to have a simple instrument that indicate it a point is stuck high or low. Most of the signals in the machine are at TTL levels and just about any logic probe will handle those, being able to monitor the 16V signals on the PMOS RAMs is very useful. However remember that even if your logic probe indicates a signal is changing, it doesn't tell you anything about the *timing* of the signal. One solution to this was HP's excellent 'LogicDart' which is a very useful instrument for debugging these machines and means a separate logic analyser is unnecessary in many cases. However this is not an easy instrument to obtain

Logic analyser : As you know, the HP9800 machines are bit-serial with most of the signals changing all the time. You need to be able to display the relative timing of several of these signals at once to trace most faults. This means a logic analyser. The machine clocks at 8MHz, so just about any analyser is fast enough, but the more channels the better. It's useful to be able to trace the processor microcode which takes 8 channels and possibly display some other signals at the same time.

Oscilloscope : This is not a particularly useful instrument for debugging the digital side of these machines, but it can be useful when repairing the analogue sections of the magnetic storage system

Bench power supply : This is by no means essential, but an adjustable DC power supply going up to at least 24V can be useful for checking motors, lamps, solenoids, etc.

Insulation tester : Although I have never found an insulation breakdown in any HP desktop calculator, you may wish to check the insulation of the mains transformer and related components for your own safety. A high-voltage insulation tester (commonly called 'Megger') is useful for this.

Again, it is worth buying good-quality test equipment. If the instruments give you misleading information you can spend a lot of time looking for faults that aren't there or conversely missing the faults that are there.

I should also point out that no piece of test equipment is a 'magic box' that will tell you just what is wrong with an HP9800 machine (or anything else for that matter). All it can do is provide you with evidence as to what is actually going on in the machine. It is up to you to interpret that evidence and find the fault.

9.4 Safety

I will credit my readers with some common sense which means this section will not contain warnings that soldering irons get hot, or that it's a bad idea to try to balance a power transformer on your nose. I shall concentrate on the less-obvious hazards not just to you, the repairer, but also to the machine itself[1].

The only real danger to the repairer is the mains supply. The outputs from the power transformer are at a fairly low voltage (although I suspect the 52V between the ends of the winding in an HP0910 or HP9820 could give a 'tingle') and are isolated from the mains. The voltages on the logic PCBs are, of course, much too low to be harmful.

The mans voltage wiring is mostly out of the way on the inside of the rear panel of the machine where it is unlikely to be a danger. However the power switch on the front of the keyboard is of course connected to the mains and is much easier to touch. When I describe how to dismantle the machine I will

[1]It is perhaps worth pointing out that HP9800 machines are a lot rarer than enthusiasts who would like to repair them...

recommend removing this switch form the keyboard chassis so that mains and logic are kept well apart.

It is almost impossible to operate this switch when it is removed from the keyboard without touching the terminals, and for this reason I do not recommend trying to use it during testing. If you have switched mains sockets on your workbench, I recommend turning the machine on and off there. Otherwise, make up a special mains lead with a double-pole switch (breaking both the live and neutral wires) and optionally a neon indicator in the middle. Make sure the earth wire is connected straight through, of course. Use that switch to turn the machine on and off for testing.

Even with the mains switch turned off or unplugged at its in-line connector, many of the components on the rear panel are still connected to the mains. Therefore, unplug the mains lead from the input connector before working on any part of the rear panel.

The main hazards to the machine also relate to high (or at least higher-than-normal) voltages. Although the power supply has crowbar protection circuits, it is well worth checking the supply before using it to power the logic boards and I will describe the procedure later on.

Another source of high voltage is, of course static electricity. The TTL logic ICs used in the HP9800s are not likely to be damaged in this way, but the ROMs and the PMOS RAMs can be, and of course these ICs are difficult to replace (the ROMs being custom-programmed for the machine and the RAMs being long out of production). When you remove a memory board from the machine. put it in an anti-static (conductive) bag. Use an earthed soldering iron when replacing components on these PCBs, and if possible work at an anti-static workstation. It is probably a good idea to avoid stroking your cat when repairing the HP9800 memory system[2].

9.5 Practical hints

9.5.1 Don't mix up the screws

When you take one of these machines apart, particularly when you come to work on the mechanical sections you will soon have quite a number of nuts, bolts, washers, clips and the like. Purchase one of those plastic boxes with many compartments – the older ones with fixed partitions are best, I've found the movable partitions tend to come loose at the worst possible time – and put associated hardware, for example the rear panel fixings, or the parts from the printer motor, in a particular compartment. If you still think

[2]What? You don't have a cat? No wonder you are having problems with your HP9800

you may forget what went where, you can put a little slip of paper in the compartment to label it.

9.5.2 Finding your way around

When you first look inside an HP9800 you'll see a mass of PCBs and will wonder how you will ever find anything.

The PCB ejector handles are colour-coded. The guides for the PCB are the same colours as the handles which simplifies getting the PCBs back in the right places. And the colours give the last 2 digits of the PCB part number using the resistor colour code. There is one exception to this code. The older HP9810 display PCB is an 09810-66541 while the later one is 09810-66542. Both of them have yellow and brown ejector handles (and the guides are yellow and brown in all machines). Since the 2 PCBs are entirely compatible, it made more sense to keep the handles matching the guides.

Directional arrows such as 'rear' or 'LHS' on the schematics assume the machine is in the normal operating position in front of you.

The ICs on each PCB are numbered as follows. Looking at the PCB, and with the edge connector fingers pointing down, IC1 is the one nearest the top left corner. The numbers then increase to the right along the top row of ICs, then carry on with the left most IC in the second row and so on. If in doubt, check that the IC number is the one you expect or even trace a few of its connections.

9.5.3 Managing without extender boards

The plug-in PCBs in an HP9800 machine are so close together that it is impossible in most cases to conenct probes to IC pins when the PCB in in the machine. Of course some signals are brought out on the test connectors, and some IC pins, particualrly those near the top edge of the PCBs, but there are many importat signls your can't get to.

While a set of exender boards would be very useful, it is possible to manange without them by removing the PCB under investigtion and soldering short lengths of wire to the connections of interest. A peice of multi-core cable or 'rainbow' ribbon cable about 10cm-15cm long and dismantled into its individual wires is ideal for this. Put the PCB back in place and connect test equipment to the free ends of (some of) the wires. Make sure the wires do not 'short' to anything they shouldn't, including the chassis of the machine. Turn on the machine and make the measurements, then turn it off, and based on the results of the measurements resolder the wires to new locations to check more signals I generally find that 2 or 3 attempts will locate

the fault.

9.5.4 Expander signals are not TTL level

The signals between the expander U2, and the And-Or-Invert gate U3 on the **Data Path** PCB are not at normal TTL levels, they are the inputs to the 'totem pole' output stage of U3. They cannot easily be monitored, and shorting the inerted signal to +5V will damage U3. This is fairly clear from the data sheets on these devices, but you may not think to check this.

9.5.5 Make notes

I find a notebook to be very useful when repairing such a machine. Record anything that's not obvious (such as the order of parts and washers on a bolt, wiring colours if you have to desolder anything, and so on). Also record any measurements (voltages, etc) if you want to be able to refer to them later.

I suppose you could use a portable computer for this, but a (paper) notebook seems more convenient.

Chapter 10

Dismantling the Machine

It is now time to take the machine apart. I shall begin by describing how to dismantle the machine into its major modules (since the HP9810 and HP90280 have very similar construction, they will be described together) and then how to dismantle the keyboard and rear panel (these sections are very similar in all the machines). A final section explains how to dismantle the optional ROM modules.

10.1 Dismantling the HP9810 and HP9820

Start by removing any ROM modules and opening the printer cover. Remove the 4 countersunk screws under it. These retain the top plate of the machine which can now be bowed up by pulling up on the printer cover and sprung out of the side plates. Put the top cover aside.

Next remove the keyboard which is anchored by 4 screws under the machine. Stand the machine on its back and remove the 2 screws right at the front edge and then the 2 further back. The former tap into metal bushes in the relatively weak plastic front strip, the latter into metal posts on the keyboard chassis. By removing them in this order, the plastic parts are not subjected to undue stress. Put the machine the right way up again and lift the keyboard assembly to free it from the printer paper guide. Unplug the keyboard cable from the **Backplane** under the front of the printer, then unplug the in-line connector leading to the mains switch and put the keyboard assembly aside for the moment.

Before removing anything further, look at the layout of the machine. On the left is the memory box containing the memory and memory control PCBs. Moving to the right, we see the 4 CPU PCBs. The peripheral section is to the right of that, containing the printer and **Card Reader Interface**

PCB, with the **I/O Backplane** behind them. Finally, on the right are the 3 power supply PCBs under a metal heatsink cover. The **Display** PCB is at the front on the left side of the machine, the card reader mechanism is in front of the power supply.

As you dismantle the machine, look for 'obvious' problems such as burnt, broken or missing components. It is quite likely that you will find a crumbly powder inside the machine, which is the remains of the original card reader drive roller. You may also find a sticky black substance on the **Backplane** at the front of the printer, which is the remains of the platen roller. Repairing these mechanical problems will be covered later.

You can dismantle each section independently of the others, but let's start at the left and work rightwards. On top of the memory box are 6 screws. Remove the 4 along the front and rear edges, leaving the 2 at the left side. Using the ejector handles, free the **Memory Backplane** from its edge connectors and lift out the memory box. Remove the remaining 2 screws on top and the corresponding pair on the bottom and remove the side cover. Take the memory PCBs out of the memory box. You may wish to put these PCBs back in the memory box after inspecting them, if not, then store the ROM and RAM PCBs in anti-static bags.

Back on the main chassis, remove the 4 CPU PCBs, the **Display** PCB and the **I/O backplane**. Unplug the edge connector at the front of the **Card Reader Interface** PCB and unplug this PCB from the **Backplane.** Free the stud from the power supply casing and lift the PCB out**.** Unplug the head connector (a little PCB with 5 pins that plugs into the **Card Reader Interface** PCB) to completely free the Card Reader Interface PCB. Remove the 4 fixing screws and take out the complete printer unit.

The heatsink on top of the power supply area has 7 screws on it. First remove the 4 pan-head ones (these screw into heatsink brackets on the PCBs). Then remove the 3 countersunk screws at the right side and take off the cover. Remove the 3 PSU PCBs; the **+/-12V PSU** PCB must be removed before the **+16V/+20V/+24V PSU** PCB.

Remove the 4 fixing screws and lift out the card reader mechanism.

The next section to remove is the complete rear panel. Unscrew the 4 screws holding the chassis-mounted edge connectors inside the power supply area of the machine; these carry the transformer secondary winding connections. Loosen the 11/32" nut at each side that holds the rear panel to the side panel, then remove the 3 countersunk screws on the underside of the machine at the rear edge. Pull the panel off rearwards, feeding the switch cable through the cardcage metalwork.

You may not have to dismantle the machine any further, but if you wish to remove the **Backplane** for cleaning or repair, continue as follows. Unplug

the **ROM Module Backplane** cable from the left edge of the **Backplane.** Remove the 2 screws holding the ROM module hosing to the cardcage and the 2 nuts holding it to the left side panel. Take out the ROM module housing and backplane, if you wish to remove the backplane from the housing, it is held on by 5 screws.

Next undo the 2 screws holding the 2 sections of the cardcage together. Loosen the nuts holding the left section to th left side panel and slide the T-bolts along the slots. Remove the screws holding this section to the base of the machine and remove it. Remove the right hand section in the same way. Finally, remove the remaining **Backplane** retaining screws and lift it out.

After cleaning these parts, and doing any repairs that are needed here, refit the **Backplane**, card cage metalwork and **ROM Module Backplane**, but leave the other parts of the machine dismantled for the moment.

10.2 Dismantling the HP9830

Begin by removing the top cover which is held on by obvious screws, Under it you will see pair of hold-down bars over the top of the PCBs. Undo the single screw and unhook the upper hold-down bar from the PSU heatsink. Then lift out the other bar.

The keyboard assembly should be removed next. Stand the machine on its back and remove the 4 screws on the underside of the machine near the front. This is one of the few times where it is better not to remove the screws in a diagonal pattern, instead remove the 2 screws on the front edge first (these screw into metal bushes in the relatively fragile plastic front of the machine) and then remove the 3 further back (which screw into metal pillars on the keyboard chassis). With the machine the right way up, lift the keyboard up to clear the eject lever of the cassette drive and then unplug the **Keyboard Cable** from the backplane (just in front of the **Display** PCB. Unplug the in-line connector on the mains switch cable and put the complete keyboard aside.

Look at the layout of the machine. The **I/O backplane** is right at the back, with a cable connected to the top edge leading to to the rear panel connector, Of the main PCBs, the **PSU** PCB is the rearmost one, with a heatsink bracket screwed to the cardcage metalwork. In front of this are a number of PCBs taking up the full width of the machine, these contain the memory and memory control circuitry. The next area consists the **Ext ROM Selector** and **Internal Ext ROM** PCBs on the left and the CPU PCBs to the right. In front of the cardcage metalwork, the **ROM backplane** is on

the left, in the centre are the tape control and **Display Driver** PCBs, with the tape drive itself to the right. Finally at the very front is the **Display** PCB.

Remove all the PCBs from the **Main Backplane**. Most of them come out without difficulty, but 3 require comment. The printer cable must be unplugged from the top of the **I/O Backplane** before it is pulled out; there's a fixing screw in the middle of the **Display** PCB and the **PSU** PCB heatsink is fixed to the chassis by 2 screws. As you remove each PCB, inspect it for burnt, damaged or missing components, and put the ROM and RAM PCBs in anti-static bags.

Remove the **ROM backplane**, the tape drive and the tape eject lever assembly, In all cases the fixing screws are obvious,

The next part to remove is the rear panel. Start by removing the nuts holding the mains switch cable clips in place. Slide the T-bolts out of the slot. Slacken the 2 nuts that hold the rear panel to the sides of the machine. On the underside of the machine, remove the 3 screws at the rear edge, then with the machine the right way up pull the panel rearwards and unplug the secondary winding edge connector from the rear of the backplane. Remove the rear panel entirely, feeding the switch cable under the card guide metalwork.

It is not normally necessary to remove the **Main Backplane** or card guides, but if you wish to do this, either to fully clean the machine or to replace edge connectors on the backplane, continue as follows. Remove the ROM door assembly and then the side card guides which are each retained by 2 screws from the rear of the machine. There is an air deflector plate at the left side of the machine which will come off when you do this. Loosen all 8 nuts that hold the card guide metalwork to the sides of the machine and slide the T-bolts along the slots, clear of the flanges on the card guides. Spring the casing apart slightly and remove the front card guide assembly. Then remove the 2 guides for the I/O modules at the rear of the machine (5 screws each, a long screwdriver is needed for the ones holding the guides to the base of the machine). There are now no fasteners retaining the rear metalwork, spring the case apart again and remove it. Finally remove all the screws holding the **Main Backplane** in position and remove it.

It is possible to remove the **Main backplane** without removing the guides by simply removing the fixing screws and sliding it out towards the front of the machine. However, a machine in need of backplane repairs is likely to need fully cleaning too, so it's probably better to remove the guides

After cleaning the machine or repairing the **Main Backplane.** refit the latter and the card guide metalwork to the casing – and don't forget to put the T-bolts in the side slots before fitting the guides. Leave the rest of the machine apart for the moment.

10.3 Keyboard dismantling

There are 2 good reasons for dismantling the keyboard. Firstly to clean out the years of accumulated dust, coffee, and the like. And secondly to remove the mains switch so that the mains can be kept well away from the rest of the machine while you are tracing and repairing faults.

The first part to remove is the **Keyboard Encoder** PCB. Put the keyboard upside-down on your bench with the display window section of the bezel overhanging the edge. In an HP9810 or HP9820 machine, unplug the keyboard cable from the end of the encoder PCB (The HP9820's cable will be completely free now, the HP9810 one is attached to the LEDs on the keyboard) and carefully ease off the **Connector** PCB between the **Keyboard Encoder** PCB and the keyboard itself. On an HP9830, there is a single PCB which connects the **Keyboard Encoder** PCB to the **Keyboard** and also carries the cable to the **Main Backplane**; ease it off. Finally remove the screws and lift off the **Encoder** PCB.

The plastic keyboard bezel is held in place by 6 screws. Four are easy to see at the sides of the keyboard, the other 2 are reached through holes in the **Keyboard** PCB. Loosen the 4 at the sides so they have about 1/4" of play, then remove the last 2 screws. The one at the rear can be extracted using tweezers or long-nose pliers working between the keyswitches, the one at the front has to be shaken out. Remove the remaining screws and the bezel. The bezel is quite brittle now, and should be stored with care.

Back on the keyboard chassis, remove the front casing strip (6 screws) – on an HP9830, desolder the power-on indicator wires from the keyboard PCB. Undo the 2 screws and separate the mains switch from the keyboard.

The keyboard will need cleaning, they always do. On an HP9810 or HP9820 (with the 'transformer' keyboard), pull out all the keycaps and their plungers. Slide the springs off, and note that double-width keys have 2 springs, each somewhat weaker than the one used on a normal key. The normal and double-width keycaps can be pulled off their plungers for cleaning, but it is better to clean the narrow keycaps on their plungers to avoid damage. Remove the screws holding the keyboard PCB to the chassis and lift it off. Remove all the plastic key housings.

On an HP9810, there are 10 status LEDs in plastic housings on the keyboard chassis. These housings are retained by C-clips. To replace an LED, remoce the C-clip and slide the housing out of the chassis. The housing will then separate into 2 halves, exposing the LED and its connections. It should be noted that the 3 LEDs associated with the ROM module keys are in longer housings than the remainder. Therefore if several LEDs are removed at once, keep a note as to shich housing parts came from each LED.

On an HP9830, pull off all the keycaps. If you want to go further, it's necessary to desolder all the keyswitches from the PCB and then remove the PCB from the chassis (the fixing screws are easy to spot)., Then unclip all the switches from the chassis. When reassembling, it's easiest to fit the PCB to the chassis and then to clip the switches in place one at a time making sure the pins go through the holes in the PCB before soldering that switch in place.

You now have the incredibly boring job of cleaning all the keycaps and the other parts of the keyboard. Then reassemble the keyboard unit itself, but do not refit the mains switch, front casing strip, bezel or encoder PCB at this stage.

On an HP9830, this is a good time to test the power-on bulb. If it is burnt-out, solder a new 5V 60mA wired-ended bulb to the terminals on the front casing strip.

The keyboard bezel will have turned very brittle with age and it is likely to be cracked. If so, remove the display window so it will not get damaged during the bezel repairs (it is held on by small screws on the bottom flange).

The best way I have found for repairing the bezel is to put the pieces together and apply a suitable solvent (such as dichloromethane (methylene chloride), which is sold under the trade name 'Plastic Weld') along the cracks with a small brush (a natural bristle brush is best since it will not dissolve in the solvent). Then cut a piece of cotton fabric to cover the cracked area on the underside of the bezel. Put this over the cracks and 'paint' it with the solvent/ When the plastic has softened, force the fabric into it. When it has dried it should be strong enough for use, but still treat it with care.

10.4 Dismantling the rear panel

The rear panel should be dismantled to inspect the mains wiring and components and also to clean and lubricate the cooling fan. Again the procedure is similar on all models.

Begin by removing both fuseholder caps and the cartridge fuses. Inspect and test the latter. If either are blown, particularly if they are shattered or blackened, there may be a serious power supply fault.

On an HP9810 or HP9820, the mains filter is likely to be a single module. There may be a metal shield in front of it, which should be removed first. Then remove the nut and bolt holding the filter to the bracket so the filter can be moved clear.

On an HP9830, it is common to have separate filter inductor and capacitor units. Undo the 4 screws holding the front shield plate over the filter

assembly, unclip the inductor and capacitor from their Terry clips and put the plate aside. Unscrew its 4 mounting pillars.

On all types of machine, unbolt the power transformer from the rear panel. Then undo the 4 nuts at the corners of the connector plate, these screw onto studs on the metal bezel. Remove the bezel completely. the connector panel is now free, but of course it is wired to the cooling fan.

The original fans used in these machines are very easy to dismantle and repair. Take out the 4 screws on the front plate (these are not the screws holding the complete fan to the rear panel of the machine, of course). Lift off the front plate complete with the fan blades and motor. There may be an earth wire on one of the fan housing screws, if so, remove that screw. All the electrical parts can now be removed from the rear panel.

The fan motor is held to the front plate by 2 screws under the label. Feel for them with a screwdriver, then remove them and the plate. Remove the circlip and washers from the end of the fan spindle then slide the rotor/blades out from the bearings. There will be spacer washers on the other end of the spindle too, make sure you don't loose any.

Back on the rear panel, remove the fan housing and grille. There are plastic bumpers at each end of the panel, which are held on by screws, remove them. If they have deteriorated, it is probably best to leave them off and simply put the screws back in their holes.

As with the keyboard, these parts will need cleaning. Inspect the parts for damage, broken connectors, and similar faults. Then reassemble the complete back panel. When reassembling the fan, put a little light machine oil on the bearings. Check the setting of the voltage selector switches, and fit good fuses of the correct rating in the holders.

It's time for the first electrical tests. If you have an insulation tester, check the resistance between the live and neutral pins of the mains input plug and between those pins and the earth pin. If you have an insulation breakdown, suspect the mains filter components. Connect the mains switch, removed from the keyboard, to its in-line connector and turn the switch on. If you are worried about the live mains on the switch terminals coming into contact with something it shouldn't, put the switch in a little plastic bag which you can fix to the cable with a cable tie.

Now check for continuity – not a 'dead short' though – between the live and neutral pins of the mains connector. If it is open-circuit, suspect the mains switch first of all. It is easy to check if the appropriate pins on the connector are linked together with the switch is turned on. If not, then it's time to replace the switch, I have never managed to successfully repair one of these. If the switch is good, check the rear panel components including the mains transformer primary windings.

Check the insulation between the live or neutral pins and the ground pin again. If you have a breakdown this time, the most likely problem is, unfortunately, the mains transformer. But I've never had this.

Time to apply mains at last. Use the switched cable I mentioned earlier if you don't have switched mains sockets on your workbench, and if either fuse had blown hard, you might want to connect a 100W mains light bulb in series with the mains input at this point. Turn on the power. The fan should start running and the bulb (if used) should remain dark. If not, suspect a short-circuit somewhere, possibly shorted turns in the mains transformer. Fortunately, though, this is rare. Using an AC voltmeter, check the output voltages of the transformer.

Assuming everything is correct, disconnect the mains input, unplug the switch at the in-line connector, and refit the rear panel to the machine. Reconnect the switch so the transformer can be powered up again.

10.5 Dismantling a ROM module

Since the ROM modules contain only custom mask-programmed ROM chips, it is unlikely that you will be able to repair a defective one. However you may wish to dismantle a ROM module to make one good one from 2 defective ones of the same type, to repair the bias potential divider, or just for interest.

It is clear that HP never intended these modules to be dismantled. They are held together by barbed metal studs forced into the plastic housing. But they can be dismantled with little damage as follows :

Put a suitable tool, such as a flat-bladed screwdriver between the 2 halves of the case at one corner, near the stud. Heat the stud's head with a soldering iron, applying a little solder will aid the heat transfer (the solder will not stick to the stud, which is aluminium). When the plastic softens, use the screwdriver to force the casing apart at that corner, remove the soldering iron and hold it open until the stud cools.

Repeat this procedure for the other 3 studs and separate the case halves. Remove the extractor handle (if it hasn't already fallen out) and lift out the PCB noting which way round it was fitted.

Push the studs out of the case half. I find it easier to reassemble these modules with #4 * 5/8" self-tapping screws, which will screw into the plastic in place of the original studs.

Chapter 11

Electronic Troubleshooting

In may ways this section is totally unnecessary. With the theory-of-operation that I have already given and basic electronic knowledge, you should be able to trace any fault in an HP9800 machine. But a description of the basic tests will be useful. Of course I can't give a sequence of tests to precisely locate the faulty component. But what I can do is describe how to determine which section of the machine is (probably) faulty and how to further investigate that section.

The methods of troubleshooting the power supply, processor and memory sections are essentially the same for all the HP9800 machines.

This will not be a list of 'stock faults'. I could say, for example, that the reason my HP9820 had no display was that U4 on the **Memory Address** PCB had failed. While this would be true, it would also be useless. As you doubtless realise by now, the same symptoms – a totally blank display – could be caused by the failure of just about any IC on that PCB. Or on any of the other PCBs in the memory or processor sections.

I am, of course, going to assume a basic familiarity with troubleshooting techniques and the testing of components. This information is not specific to the HP9800 series and is covered in many books on faultfinding and repair.

11.1 Initial tests

At this point the machine should be in the following state. All the plug-in PCBs and the keyboard should be removed, the only parts in the case should be the **Backplane**, the cardcage metalwork, and the (tested) rear panel assembly. The mains switch should be connected to its in-line connector and turned on, Make sure the connections on this switch can't come into contact with anything – including you. Of course the machine should not be

connected to the mains at this point.

11.1.1 Power supply troubleshooting

Fit the **PSU** PCB(s), connect the mains cable and turn it on. If the cooling fan starts running, the mains fuse has not blown, and the next stage is to check the power supply output voltages. There are testpoints for all of these apart from possibly the +5V rail on the power supply PCBs, check the voltage on each of these testpoints with respect to the metal chassis. If there isn't a +5V testpoint, check this on the appropriate pin of an edge connector on the **Backplane**.

If one or more of the power supply voltages is incorrect, first remember that all the other outputs use the +12V output as a voltage reference, so this output must be put right first. Check the unregulated input voltage to the regulator, if this is missing, check the diodes and smoothing capacitor. Check the pass (or chopper) transistor. Measure the voltages on the inputs of the associated 723 IC, and based on the theory of operation of that regulator, diagnose the problem.

If the mains fuse blows when the power supply PCB(s) are fitted, it almost certainly indicates a short-circuit in the power supply section. the most likely problems are a shorted rectifier diode, a shorted smoothing capacitor[1], or a shorted pass transistor. The last will cause the crowbar circuit on that power supply output to operate, and as the pass transistor is assumed to be shorted, this will put a short across the unregulated supply, blowing the mains fuse.

If you have difficulties with the power supply, desolder all the rectifier diodes (on all the PCBs) apart from the pair that feed the +12V line. Check those last 2, the smoothing capacitor, the pass transistor and all the associated components. Get the +12V rail right first of all. Then fit th 2 diodes – after checking them – that feed the -12V regulator, and get that one working. Add the diodes for the remaining linear regulators one set at a time and get each supply working. Finally fit the diodes for the +5V switching regulator (which is the most complicated one) and troubleshoot that.

Once the power supply is working correctly, refit the power supply cover ant its screws (starting with the 3 countersunk ones at the right hand side) in an HP9810 or HP9820, or screw the **PSU** PCB heatsink to the cardcage in an HP9830. This ensures the power supply components have the correct heatsink arrangements.

[1]Actually, the electrolytic capacitors used in these machines are a lot more reliable than some people believe and rarely need replacement

11.2 System clock

One digital section can be easily tested independently of all the others and that is the system clock. With the machine turned off, fit the **Clock** PCB into its slot. Switch on and check the **MClk** signal at U5d with a logic analyser or oscilloscope – with nothing else in the backplane it is easy to connect a probe here. You should see an 8MHz square wave, if not, check the master clock components.

Sometimes you find a very poor square wave around 19MHz here. No, you don't have a faster-than-normal HP9800. What you are seeing is U5 oscillating by itself because the crystal has failed or is not connected. Check that the 8MHz crystal is properly plugged in, and if necessary replace it.

Now, again using the logic analyser or oscilloscope, monitor the **Bitclk** and μ**Clk** signals at the outputs of U13. These are (or at least should be) repetitive waveforms so a normal oscilloscope is adequate for this. You should seen 16 pulses on **Bitclk**, then a pulse on μ**Clk** then another 16 pulses on **Bitclk** and so on. If not, troubleshoot the bit counter and clock control circuitry.

If you want to do a more thorough test, you can ground the μ**(8)**...μ**(11)** signals in various combinations and check that you get 1 more **Bitclk** pulse between μ**Clk** pulses than the binary number at the inputs to U14. If not, then U14 is suspect, but this is a very uncommon fault.

11.3 Obtaining a display

Since the display in these machines is controlled by the system firmware, to get anything at all on the display, the processor, memory system and display logic itself must all be operational. Unfortunately there is no simple way to test these sections one at a time, it is simpler to assemble the complete processor and memory system and try to diagnose any faults in the system as a whole.

With the machine disconnected from the mains, fit the remaining 3 CPU PCBs, the complete memory system (apart from any RAM expansion PCBs) and the display system. For an HP9810 or HP9820, the 'memory system' consists of the memory box and the PCBs in it apart from the RAM expansion PCB in the very top connector. For an HP9830, it's the **Memory Address**,**Memory Data**, **Ext ROM Selector** and **ROM** PCBs, the extension ROM PCB in the rearmost connector and the **RAM** PCB in the connector nearest the **PSU PCB**. The display system consists of the **Display** PCB in all machines and in the case of the HP9830 the **Display Driver**

PCB too.

Switch on the machine. If you are very lucky, you will get something on the display. If you are extremely lucky you will get the correct display. But in most cases the display will remain blank.

Check the power supply voltages again. It is entirely possible that a short-circuit on one of the PCBs is causing one of the regulators to shut down. This may be due to a defective decoupling capacitor. In any case, find the short and correct it before carrying on.

Now think back to the fundamental rule of troubleshooting, and consider what the machine should be doing. At one level what it should be doing is 'executing machine instructions, some of which output data to the display'. That means the processor should be running, it should be accessing ROM and RAM, and it should be loading data into various registers (including the I/O register) and driving the display.

For the processor to be executing instructions, it means that it's reading them from memory into the Q register and that it's incrementing the program counter (using the ALU) to fetch the next instruction (most of the time, there will be jumps, of course).

First monitor the **DispStb/** signal at the display. If it is pulsing, then it is likely the processor and memory system is close to working. Monitor the I/O register outputs and if those are toggling too, the problem is most likely to be in the display logic itself. This is rarely the case, but you will feel a right idiot if you spend days working through the processor logic only to find it was a problem with a decoder on the display PCB. I shall have more to say about troubleshooting the display itself later on.

Check **Bitclk** and **μClk** again. If they are no longer running, something must be stopping them. If just **μClk** is stopped on an HP9820 or HP9830, check round the DMA flip-flop in case **DisμClk2** is being asserted. If both clock signals are missing, then check the **Wait/** input to U7a on the **Clock** PCB.

11.3.1 Is it stuck in a refresh cycle?

If **Wait/** is asserted, it is likely that the machine is stuck in the memory refresh cycle. Examine this logic on the **Memory Address** or **Memory Timing** PCB. In particular, is the refresh cycle running at all (is the memory control counter or state machine running), does the refresh address increment so that the refresh cycle should end, and if it does, why isn't **Wait/** being deasserted?

11.3.2 Ins the microcode running at all?

Since **μClk** is operating, the outputs of the microcode program counter should be changing. Check the **μAddr** signals at the test connector on the **CPU Control** PCB. At least some should be toggling. If not, see if any of the inputs to the microcode program counter flip-flops are high (these are also available on the same test connector). If they are, suspect a problem with the microcode program counter flip-flop.

As an aside at this point, if all the **μAddr** signals are toggling *apart from* **μAddr(1)**, it is very likely the microcode is stuck in the I/O loop at 0512, 1207. We will confirm this in a moment.

11.3.3 Tracing the microcode

If you have a logic analyser, one very useful technique is to trace the microcode. Connect 8 of the logic analyser channels to the **μAddr** signals on the **CPU Control** PCB test connector and if possible clock the analyser (at least for the simpler tests) from the rising edge of **μClk**.

Using the microcode source listing, see if the sequence of states is possible. If not, then the fault is most likely to be in the microcode program counter or microcode ROM areas of the machine. If the sequence of states is possible, then see if the machine ever executes the instruction fetch sequence starting at 0202. If not, then the microcode must be stuck in a loop somewhere. Determine what should get it out of that loop, and find out why it isn't. One common loop is, as I mentioned earlier, the I/O instruction loop, and I will say more about that later on.

Another possible loop is the address indirection loop at 0003, 0602. If, due to a fault, all addresses are read with their MSB set, the microcode will loop here as it repeatedly indirects on them. This generally indicates a fault with the memory or **Memory Data** PCBs. Check the logic that loads the MSB of the T register.

If you do not have a logic analyser, all it not lost. It is possible to do a simpler test with a home-made device. It won't give you as much information, in particular, it won't tell you the exact microcode sequence. But it will let you see if various microinstructions are being executed.

You need an 8 bit comparator chip such as a 74LS688. Power it from the machine's +5V supply and connect one set of inputs – say P(n) – to **μAddr** signals on the **CPU Control** PCB test connector. Connect the other inputs of the comparator (Q(n)) to a suitable switch (an 8 position DIP switch, for example) with pull-up resistors. Ground the enable input on the comparator and monitor the output with some device that will detect a narrow pulse such

as a logic probe. By setting the switches it is possible to see if particular microinstructions are being executed,

For example, see if instructions are being set by setting the Q inputs of the comparator to 1612 (11101010_2). Or see if the processor ever enters the I/O loop by setting 0512 (01011010_2) and if it leaves it by setting 0513 (01011011_2). Of course if you are using a switch to ground with pull-up resistors, a closed switch represents a logic 0.

In this way it is generally possible to get a good idea as to what the processor is doing with quite simple equipment.

11.3.4 Is the processor attempting to read instructions from memory?

The first part of the instruction fetch at microcode locations 0202 and 1603 increments the P register (machine code program counter and also loads the increments version in the M register (memory address register). If those microinstructions are being executed, the clock input to U10 on the **Data Path** PCB should be active. **Pout** should be toggling as the P register is read into the ALU. If you have a logic analyser, or some other way to check the relative timing of signals you could check that the data is getting to the ALU via the data selector U3, and that the ALU output is being transfered back to the P register. For these tests, where you are looking at the processor behaviour during a microinstruction, the analyser should be clocked by **MClk**

In any case the binary carry flag, U12b, on the **Data Path** PCB should be toggling. Without a logic analyser you can't tell if this is happening at the right time, but if it's 'stuck' then that area needs investigation.

Since the M register should be loaded by these microinstructions, its clock (on the Clk1 inputs of the shift register ICs) should be active. If not then there is a problem with the clock control circuitry on the **Memory Address** PCB. Now look at the **M** outputs of the M register. As the new address is shifted into the register, all of them should be changing. If they are stuck after a certainly point, then there is a problem with the M register. But take care. If, for example **M(7)** and all lower bits are stuck low, you may think the problem has to be in the shift register IC that produces **M(7)...M(4)**. However, if the next *higher* shift register chip is malfunctioning, the timing of the serial input to the suspected one may be incorrect and this could be the cause of the fault[2]. So check carefully before replacing any ICs.

[2]If this sounds like the voice of experience, you are right.

11.3.5 Is the memory controller running?

The microinstruction at location 1612 reads the next machine instruction from memory. It therefore asserts **R(6)**, so check this signal is going low. This, in turn should assert **MemEn**, which should be checked.

If so then the memory control counter or state machine should be running (again, check it). A logic analyser will let you check the exact sequence of the control signals (and this is almost essential to trace faults within the state-machine based memory controller in the HP9830), but even a logic probe will allow you to see if the machine is trying to access memory

It is extremely likely that the processor will be trying to write to the RAM too. Check if **R(3)** is ever going low, if so, then check that there is some evidence of RAM writing, such as the the write enable input to the RAMs being asserted.

If you have a logic analyser, you can check the timing of all the memory control signals against the theory-of-operation of this section. This is necessary if simple tests don't find the fault.

11.3.6 is the memory being addressed?

Since the M register is being loaded and the memory controller is running, there should be activity on the memory address lines. Check this. In particular check the level shifters to the RAM address inputs.

Check the address decoding circuitry to find out which memory device(s) should be being accessed. In particular, make sure a ROM is being accessed, and if memory write operations are taking place that RAM is being accessed too. Again, check the level shifters for these signals, they can cause problems. Check the logic that provides the A(8) and A(8)/ enable inputs to the ROMs too.

If all this appears to be working, the selected ROM should be outputting something. Check it is. Of course it is unlikely you will know just what the data should be, but if any data line appears 'stuck', then it needs further investigation. If RAM is being accessed, check that the RAM output level shifters are working correctly.

Of course the final stage of a memory read operation is to load the data into the T register. Check there is activity on the data inputs of these shift register chips and that the mode and clock inputs are causing it to parallel-load. You should also, of course see activity on all the outputs of the T register both as it is loaded and as it recirculates to read the data into the processor. Only a logic analyser can distinguish between these, but a logic probe should at least show activity on all the T register outputs.

11.3.7 is the instruction being read and decoded?

The next microinstruction, at location 0616, transfers the machine instruction from the T register to the Q register. Check that this microinstruction is being executed. Then check that the T register is recirculating as it is read. The mode input to these shift register ICs should be low and the clk1 inputs should be toggling at this time. With a logic analyser you can verify this exactly, with a logic probe you should see activity on both these signals (the mode input would be high during the read cycle and go low for the recirculation, so you should see it changing state with a logic probe).

Of course during recirculation the data from the LSB of the T register is fed back to the serial input. This is something that is very difficult to check without a logic analyser, since data from the processor will be being loaded into the T register before RAM write operations. But if nothing is happening at the serial input of the T register, there is certainly a fault.

Check that the appropriate gate in the ALU input data selector is being enabled (Again, a logic analyser will also check this this is happening at the right time). The data should be passing through the ALU to the input of the Q register. A logic probe on its own will tell you almost nothing here (the ALU output is changing, sure, but then it would be changing in most other microinstructions too).

Now, the Q register clock input, from U11c on the **CPU Control** PCB,should be active to load the data into the Q register, so check this next. And as with the other shift registers, check that all the outputs of the Q register are toggling as the instruction is loaded it.

At some point a memory reference instruction will be executed, and after calculating the address the microcode will end up at location 0602. There is little point in checking the output of U4 on the **CPU Control** PCB with a logic probe, since it should be changing as the Q register is loaded, but if it's stuck, it is probably worth checking this gate. It's likely any problems in the microcode branch condition logic would have been found by now , but it may be worth checking this too.

At location 0602 there is the multi-way QμJmp operation, Check that **QμJmp/** is being taken low, if not, debug round U15b on the **Data Path** PCB. The actual multi-way jump, controlled by U15 on the **CPU Control** PCB is very difficult to trace without a logic analyser, and while U15 must be doing something for the microcode to get this far, a fault could affect QμJmp operations only, although this is fairly unlikely.

Note that if the Q register is always 0, either because the T register is not being correctly loaded from memory during the instruction fetch, so that it ends up containing 0, or because the T register is not correctly transferred to

the Q register due to data path problems, then the processor will repeatedly execute ADA instructions. That is, the microcode will execute the routine at 0000, 0005, etc after every fetch. If this appears to be occurring, check to see if the T and Q registers are being loaded correctly

11.3.8 Are instructions being executed correctly?

This, essentially, means 'Is the data path – the ALU and registers – working correctly?'. Some faults in this area will already have been detected, for example if the ALU output was stuck high or low, the M register could not be loaded correctly.

Without a logic analyser, again all you can really do is test that things are doing something, For example, when the microinstruction at 1213 is executed, as it should be during the instruction decode routine, the accumulator selection flip-flop, U2 on the **CPU Control** PCB is set from bit 11 of the Q register. It is very likely, therefore, that this flip-flop should be changing state as different machine instructions are executed. A logic probe will tell you that, a logic analyser will allow you to check it's actually doing the right thing.

With a logic analyser you can trigger on a particular machine instruction in the Q register and check that the right accumulator control signals are produced during the microinstructions for that machine instruction. Without one, you can only see if the accumulators are doing *something*. Actually, this is often enough, most common faults cause a signal to be totally absent.

11.3.9 Are any I/O operations occurring, and are they terminating

If the microcode is running normally. see if it ever gets to location 0217 which is the start of the I/O routine. If not, then either there's a problem with the instruction decode routines (possibly in the microcode conditional branch logic) or genuinely no I/O instructions are being read by the processor. This may be due to a (very rare, fortunately) problem with the system ROMs.

Assuming the I/O routine is being entered, check the output of the I/O Operation flip-flip, U1, on the **Clock** PCB. **I/OOp** should be asserted at the start of I/O operations and deasserted at the end.

If it is never deasserted, the microcode will sit in the I/O loop already mentioned for ever. If the microcode is stuck in this loop but **I/OOp** is not asserted, check the microcode conditional branch logic, in particular **Cond(15)**.

If the microcode is still stuck in the I.O loop with **I/OOp** asserted, check the outputs of the I/O Cmd Decoder, U9 on the **Clock** PCB. Hopefully it will be stuck in one of the data transfer operations with one of outputs 10, 12 or 14 of U9 pulsing. If so, check if U4 is counting down, and if it is, whether **I/OBr** is ever being asserted. If U4 is not behaving correctly, then check it, and its inputs. If U4 is working properly, or if the I/O control system is stuck performing a non-data-transfer operation (which should complete in a couple of cycles), investigate the I/O End logic at the inputs to the I/O Cmd Register.

If the I/O microcode is apparently running normally, that is, if the I/O loop is being entered and exited normally, check if any output instruction are being executed. This can be easily done by seeing if output 12 of U9 on the **Clock** PCB is ever being asserted. Again, if not, there may be a firmware problem. If it is, see if the I/O register on the **I/O Interface** PCB is receiving data from U13b on that PCB, and if that data is being shifted along the register. A logic probe applied to all 16 outputs in turn (these are accessible on the edge connector for the **I/O Backplane** should show them all toggling.

finally, if you still have no display, investigate the Display Strobe flip flop consisting of U2c and U7a on the **I/O Interface** PCB.

By now you should have something on the display. The next task is to find any faults in the display logic itself.

11.4 Obtaining the right display

Once something appears in the display, the processor and memory sections must be basically working. However, the display could still be incorrect. You may find it useful to read the next section and get the keyboard working so that you can enter data into the display for some tests.

There are several cases to consider :

11.4.1 The display is blank, even though the processor is accessing it

If you have a logic analyser, capture the contents of the I/O register when **DispStb** is active, and check that the machine is outputting something that should result in a display. If not, then this would suggest a RAM or firmware ROM problem.

If valid display data is being sent to the display, check the display protection monostable circuit. In an HP9810, check the cathode driver decoders

(which are common to all 3 rows of the display). In an HP9820 check the row group selection circuitry.

In an HP9830, check that the display scanning oscillator, U6, on the **Display Driver** PCB is running, and that the scan counter (U12) is incrementing. Check that the outputs of the character generator ROM are active. Also check the column select decoder, U35, and the display enable logic on the **Display** PCB itself.

11.4.2 The display shows the wrong characters or meaningless patterns

Since the displays on the HP9810 or HP9820 are firmware-driven, this would suggest a problem with the firmware ROMs or system RAM on these machines.

In an HP9830, this could also be due to problems with the character generator ROM and its input multiplexer on the **Display Driver** PCB.

11.4.3 A complete (electrical) row or column is blank

It is important to remember that on the HP9820 and HP9830 machines, the electrical matrix of LEDs is not the same as the physical 7×80 or 7×160 matrix of LEDs.

If all the LEDs in a particular electrical row or column are blank, then of course attention should be turned to the driver circuit for that row or column and the logic circuitry that feeds that driver. If several rows or columns are blank, look for a common cause. For example, an HP9810 where the 'a', 'b', 'f' and 'g' segments of the 'Y' register display are all dark could mean that U3 on the **Display** PCB has completely failed.

11.4.4 One LED never lights

If one segment of an HP9810 display or one dot of an HP9820 or HP9830 display is always dark and the display otherwise operates normally, it would suggest that that particular LED has failed.

The LEDs can be tested using an analogue multimeter on the '$\Omega \times 1$' range with the probes applied to the pins of the display device directly[3], the machine being turned off and the **Display** PCB removed.

The remedy is obvious: replace the defective display device. The problem is obtaining one. The 7-segment LED modules used in the later version of the

[3]With most analogue multimeters, the black probe is *positive* in the resistance mode

HP9810 **Display** PCB are similar to those used in 'classic series' handheld calculators, and the latter may be a source of spares. But the other display devices are very difficult to find.

11.5 Repairing the keyboard

The next stage is to assemble and connect up the keyboard. Fit the **Keyboard Encoder** PCB to the underside of the **Keyboard** with 2 of its screws (you will have to remove it again later) and plug in the interconnection PCB and, on an HP9810 or HP9830, the cable. Connect the keyboard cable to the appropriate connector on the **Backplane** and turn on the machine. Try the keys. If you are lucky, they will all operate correctly, if not...

11.5.1 The keyboard seems dead

If no keys have any effect, then monitor **KeyDn/** with a logic probe. This signal should pulse low when a key is pressed. On an HP9810, it should stay low while the key is down, on the other machines it may well be set high again by the interrupt service routine setting a flip-flop on the **Keyboard Encoder** PCB.

If **KeyDn** is active, then trace the processor microcode. The microcode interrupt service routine starts at location 1212. If this is being executed correctly, it would suggest that either there's a firmware problem or that the keyboard is sending invalid keycodes. Check the keyboard output gates and that the keycode is being correctly transferred to the I/O register by the logic around U6a on the **I/O Interface** PCB.

If **KeyDn/** is not active, the problem is in the keyboard. Check that the master clock oscillator is running, and that the scan counter is incrementing. Check the column driver, and in an HP9810 or HP9820, the row driver. See if there is any activity on the **Key** signal when a key is pressed. If not, on an HP9830, check the row multiplexer, U3, on the **Keyboard Encoder** PCB. On an HP9180 or HP91820, check the key sense comparator and its associated components.

If **Key** is active, check that it is stopping the scan counter, and check the logic that generates **KeyDn/**

11.5.2 Only some keys work

If the defective keys all lie in the same row or column of the keyboard matrix, the most likely problem is the driver circuit for that row or column. This

should be relatively easy to repair.

If the 2 keys at the same matrix location on an HP9810 or HP920 are defective and all others work, check the diode on the **Keyboard** PCB associated with that location.

If half the keys on an HP9810 or HP920 fail to work, it is likely they all have keycodes with the LSB in the same state, and that one half the key sense comparator, or its associated components, has failed. The main difficulty here is obtaining a replacement comparator IC.

One curious fault is when a random set of keys on an HP9810 or HP9820 fail, all with the same state of the LSB, indicating a problem with one half of the comparator, but other keys using that half work correctly. This is normally due to the fact that the signals from the various keys are not all identical in voltage so if the comparator circuit is malfunctioning it may detect some and not others. The cure is to replace the comparator IC, but since this is difficult to obtain, a temporary fix is to alter the values of the resistors associated with its inputs to increase the sensitivity slightly.

Sometimes this fault occurs on both halves of the comparator at the same time. Then a seemingly random set of keys fails, quite often one key at a particular matrix location works, the other doesn't. The remedy is as I have just described.

An individual key not working on an HP9830 keyboard generally indicates a problem with that particular keyswitch. This can be easily verified removing the **Keyboard Encoder** PCB again and connecting an ohmmeter across the connections to the suspect switch on the **Keyboard** PCB. If it doesn't operate correctly, remove its keycap and the surrounding keycaps, desolder the switch form the PCB and unclip it from the keyboard chassis.

The HP9830 keyswitches were actually a standard part from Cherry at the time the machine was made, but they haven't been produced for many years and are not easy to obtain now. You may have some success in spraying propan-2-ol into the switch around the plunger, or you may be able to crack apart the 2 sections of the body, clean the contacts and glue the switch together again using a suitable solvent.

One tip if you only have a few defective keyswitches on an HP9830 is to exchange them with little-used keys, for example the numeric keypad (which is wired in parallel with the corresponding keys on the main keyboard and is thus not essential to using the machine).

The Shift keys on the HP9830 are not part of the main keyboard matrix. If they fail to work, check the switches themselves and U16f, U7a on the **Keyboard Encoder** PCB. The rewind switch can only be tested once the tape system is reassembled and working, but it is, of course, possible to check that its contacts close at this stage.

11.5.3 Some keys have the wrong effect

This may, of course, be due to a problem with the firmware routines that handle those keys. But it can also be due to a hardware fault in the keyboard itself.

For each of the faulty keys determine its keycode and the keycode it appears to be acting as using the schematic of the **Keyboard** PCB.

If the 2 keycodes for all the faulty keys differ in the same bit, it is likely that the fault is is the output gate that transfers that bit from the scan counter to the I/O register, or a fault with the I/O register itself.

11.5.4 The HP9810 keyboard LEDs

The LEDs on the HP9810 keyboard are vary easy to troubleshoot. The LEDs themselves can be tested with an analogue ohmmeter connected to the contacts on the keyboard cable connector (unplugged from the **Keyboard Encoder** PCB, as described above for testing the display LEDs.

If an individual LED doesn't light when it should do, and the LED itself is working, then the most likely problem is the latch that drives it or the inverter that the input of that latch.

If the LEDs never change, check if **LEDStb** is ever active, if not, investigate the logic around U10 on the **I/O Interface** PCB.

11.6 Fitting the I/O backplane and expansion RAM PCBs

Now that the machine is basically working, fit the **I/O backplane** and test it again. If it fails to work, on an HP9810 or HP9820 the problem can only be a short-circuit between tracks on this PCB. On an HP9830, it is possible for the printer interface logic to corrupt the I/O bus, but this is very rare. It should not be difficult to find the fault if, say, **DO(2)** is being forced low due to a short-circuit in U4.

Next fit the **Expansion RAM** PCB if you have one. Test the machine again and check that the RAM is recognised. A fault here could be due to a defective RAM IC on the **Expansion RAM** PCB, or to a fault in the address decoder logic on the **Memory Address** PCB.

On an HP9830, refit the **ROM backplane.** A failure of one of the buffer ICs on this PCB could stop the machine from working by corrupting the **Dout(n)** memory data bus. But this is normally a trouble-free section.

Chapter 12

Repairing the HP9810/HP9820 Card Reader

12.1 Overhauling the card reader mechanism

Most of the faults in the card reader are mechanical. The card reader drive roller will certainly need replacement unless this has been done recently, and the drive belt may be defective

The card reader mechanism consists of 2 side plates joined by pillars. Plastic card guides are riveted to these side plates and these also hold the lampholders and photodetectors for the card sensors. A motor with integral gearbox is fixed to the left side plate, this drives the card roller spindle via a drive belt on the outside of the reader. The magnetic head is fixed in a metal block bolted to the inside of the right hand side plate. A pressure roller assembly runs in bushes moulded into the card guides and holds the card against the drive roller. A further 'centre' card guide is carried on the lower front pillar.

12.1.1 Dismantling

With the card reader mechanism removed from the machine, begin by removing the drive belt by simply stretching it over the pulleys. Then remove the 2 screws holding the head mounting block in place and remove the hard assembly. Put this carefully aside. It is, of course, a good idea to clean the head face with propan-2-ol while it is out of the reader.

Remove the 2 lampholders from the card guide on the left side plate by simply pulling them out of their mountings. Remove the screw on the right hand end of the lower front pillar, and the screws from the remaining 4 pillars on the left side. Start to separate the plates. Slide the small PCBs carrying

the photodetectors out of their slots on the left card guide, taking great care not to break the guide. Separate the plates and remove the pressure roller assembly. Remove the screw on the lower front pillar on the left side plate and remove this pillar with the centre card guide.

To fully separate the plates it is necessary to disconnect the wires from the drive motor. Loosen the 2 small screws that retain the rear end cap on the motor and ease it off Make a note of how the wires are connected and then remove the terminal screws. Put the right hand side plate with the wiring harness aside.

You may want to remove the brushes and clean the commutator at this stage. The brushes are retained by one further screw each, the plastic cover that acts as a support for the brushgear is held on by the 2 screws that retained the rear end cap and one further screw. Removing these parts should not pose any problems. Clean the motor parts, the pressure roller, etc in the normal way.

12.1.2 Replacing the roller

It is now necessary to remove the old roller hub from the spindle. Clean off the remains of the 'tyre' and attempt to remove the roller from the spindle using the puller tool described earlier. Fit the base plate of the puller between the roller hubs and the left side plate with the puller bolt pressing on the right hand end of the spindle. Tighten the bolt and the roller should be drawn off. Remove the old roller hubs and slide the spindle and its earthing bracket out of the left side plate. Clean the dismantled parts of the reader.

It is now necessary to make a replacement roller hub. and I am afraid to say that this requires a small lathe. Start with a length of nominal 7/8" brass rod about 1" long and measure its diameter accurately.. Put it in the 3-jaw chuck and face off the end. Using a centre drill followed by a twist drill in the tailstock chuck, drill a 3.2mm hole along the axis of this rod. Check that tithe roller spindle will slide into this hole.

You now need to machine 2 grooves into the brass to take O-rings so that the diameter at the base of the groove is 3/4". Calculate the depth from the diameter you measured earlier. Put the centre of one of the grooves 0.1" from the end of the rod, and the other a further 0.15" along. I have found that the brazed-tip round-nose tool available from many model engineering suppliers[1] makes an ideal form tool to cut these grooves. Fit a BS210 O-ring[2] into each groove in turn and check that the outside diameter of the fitted

[1]For example GLR Distributors order code 131160

[2]That's 1" outside diameter, 3/4" inside diameter, 1/8" thick

O-ring is 1".

Part off a total of 0.35" of the rod. This will result in a symmetrical component with a groove 0.1" from each end. Drill a radial hole, 2.1mm in diameter at the base of one of the grooves and tap it to take an M2.5 set screw. As a practical tip at this point, it is easier to use hex socket ('Allen') setscrews since they can be held on the end of the hex driver ('Allen key') tool and fitted into position.

Temporarily refit the spindle to the left side plate of the reader and fit the new hub onto it. Fit the right hand side plate, position the hub and mark its position on the spindle. Dismantle the reader again and mill or file a flat on the spindle to take the end of the set screw.

In a few cases the roller is firmly bonded to the spindle and the puller bends the latter before the roller comes free. In this case it is necessary to make a new spindle as well as the roller hub. Remove the belt pulley from the end of the old spindle, for example by supporting the side plate on a bench vice and driving the spindle out with a hammer and pin punch.

The replacement spindle is made from 1/8" stainless steel rod. One end has to be fitted to the pulley. The end of this rod should be turned down in the lathe to be a sliding fit in the bore of the pulley.

A metal pulley can be drilled and tapped radially for an M2.5 setscrews and a flat milled or filed at the end of the rod for this to locate on. If your M2.5 tape is not long enough to tap this hole to the full depth, tap it as deep as you can and put a small metal rod under the setscrew to press against the flat.

If the pulley is plastic, it is easier to drill and tap the new spindle axially for an M2 screw. Fit the spindle onto the end of the new spindle (it won't go on too far as the end was turned to fit in the pulley bore) and anchor the pulley with an M2 pan head screw and shakeproof washer.

Temporarily reassemble the reader side plates and insert the new spindle in its bearing holes. Mark the required length just beyond the right hand side plate, remove the spindle, cut it to length and face off the end.

Once the new spindle has been made, fit it to the side plate and the new hub as described above, mark the position of the roller and mill or file a slot for the setscrew.

12.1.3 Reassembly and electrical tests

Fit the spindle through the earthing strip and left side plate and fit the new hub, locking it in place with the setscrew. Fit a pair of BS210 O-rings to the grooves in the new hub.

Reassemble the motor brush gear carrier and the brushes. Before connecting the wiring or fitting the rear end cap, connect a 24V power supply between the brushes. The motor should run, if not, check that the brushes are located correctly.

Reconnect the wiring, fit the motor end cap and reassemble the card guides, pressure roller and side plates. You will notice that the motor wires are connected to a tag strip on the inside of the right hand side plate which carries the anti-back-emf protection diode. Connect the 24V power supply to these tags, taking care to ensure that the cathode (banded end) of the diode is connected to the positive side of the supply. Again the motor should run.

Fit the drive belt by stretching it over the pulleys and try again. The drive roller should rotate. Feed in a card. If it doesn't pass correctly through the reader, check to see what is slipping. Most of the time the problem is the drive belt on the left side of the reader.

There are 2 ways to determine the length of this drive belt. A calculator enthusiast would measure the diameter of the 2 pulleys and the distance between their centres with callipers and enter the data into the Belt Length program of the HP67 M.E. Pac 1 (Program ME1-17A) or a similar program for the HP41. A practical person would run a length of string round the pulleys, mark it, remove it and measure the length between the marks. Of course, once the length is known, it is as easy as (dividing by) π to find the diameter of the O-ring that would make a suitable belt.

Being both a calculator enthusiast and a practical person, I did both. And they agreed to well within the accuracy of the measurements. This suggested that a BS231 O-ring[3] would be an ideal drive belt for this reader, and tests showed this was correct.

With the drive system working, it is worth checking the 4 card sensor lamps by measuring the resistance from the lampholder body to the end contact. If any are open-circuit, pull them out of the card guide and dismantle the lampholder by unscrewing the 5/16" nut on the front end. This will also unscrew the central tube and release the bulb. The latter is a T1 midget flange bulb rated at 6V, 50mA, and is still available[4].

It is worth checking the head windings for continuity at this point by measuring the resistance between the centre pin and each of the 4 corner pins of the connector 'plug' (actually a small PCB with 5 pins soldered to it). Use an ohmmeter which only supplies a small test current (use it on a high resistance range if in doubt), and if possible demagnetise the head with a 'defluxer' afterwards. I suspect that replacement heads are impossible to

[3]Outside diameter = 2+7/8", inside diameter = 2+5/8", thickness = 1/8"

[4]For example RS Components 360-7531.

obtain for these machines, though.

Finally refit the magnetic head and refit the card reader mechanism and the **Card Reader Interface** to the machine.

12.2 Troubleshooting the card reader electronics

Most of the (rare)failures that occur on the **Card Reader Interface** PCB are in the motor driver circuit.

12.2.1 Motor driver

With the card reader system connected to the rest of the machine, perform any load or store operation. If the card reader motor does not start, first check that U5a on the **Card Reader Interface** PCB is being set. If not, check the **CROut** signal from the **I/O Interface** PCB and the outputs of the I/O register. Then check the 2-transistor circuit connected to the output of this flip-flop. The most common problem is for the PNP transistor that actually feeds the motor to fail.

12.2.2 Card sensors

Next check the card sensors. With the motor running (and thus **SenseEn** at +12V), check that the appropriate **DI(n)** signal changes state as each light beam is interrupted. Of course the most common problem here is the failure of the filament lamp in the card reader mechanism, but you should have already checked these. In any case, a quick look at the reader mechanism when the machine is switched on will detect this fault.

12.2.3 Writing

The next test it to type in a short test program and attempt to save it on a blank card. Using your logic probe, check that U8a's output on the **Card Reader Interface** PCB is low throughout writing, and that there is activity on the outputs of the 4 flip-flops driving the write drives. If you have an oscilloscope, check for activity on the collectors of the driver transistors and the head connections. You may have to write the program several times to be able to check all the signals.

Faults are normally limited to one channel, and are thus fairly easy to find. If the 'S' timing channel is malfunctioning, also check the **CRSout** signal from the **I/O Interface** PCB.

12.2.4 Reading

Now try to re-load the program you just saved. If it loads without errors, there is nothing more to be done, if not, then start by checking that U8a on the **Card Reader Interface** PCB is set (to disable writing) throughout the read operation. Also check that the outputs of U9e, U9f, U7f and U7e are low.

Next check that **ClRd/**. for example on the reset input of U10b, is high when the card is actually passing through the reader. Look for pulses on **CRRdClk**, if this signal seems dead, trace the signal from the head through the 'S' channel read amplifier with an oscilloscope. Then look for activity on the outputs of the read flip-flops, U10b, U10a and U11b. If any are static, investigate that channel's read amplifier.

It is, of course, possible that the reader is working correctly, but that **I/OFlg/** is never being asserted so that the processor ignores the incoming data. Check the SR flip-flop U4c and U4d and the data selector U9 on the **I/O Interface** PCB.

Chapter 13

Repairing the HP9810/HP9820 Printer

13.1 Overhauling the printer mechanism

As a mechanical device, the internal printer of the HP9810 and HP9820 machines benefits from being dismantled and cleaned. The platen roller may also have decayed with time and turned into a black, sticky, goo and this clearly needs to be cleaned off and the roller replaced.

13.1.1 Dismantling the printer mechanism

Being by removing the 3 screws holding the **Printer Interface** PCB to the right-hand side of the printer chassis. Hold the **Printer Driver** PCB in place on the underside of the printer and ease the **Printer Interface** PCB off, separating the edge connector from the card edge of the **Printer Driver** PCB. Inspect the **Printer Interface** PCB for burnt r damaged components and set it aside.

At the front of the printer, there is a metal beam running across the chassis in front of the printhead. In the middle of this beam is a hexagonal domed housing that contains the printhead pressure spring. Unscrew this with a 5/16" nutdriver – carefully so the spring doesn't escape – and remove it and the spring. Undo the 2 screws that hold the beam to the bearing plates (noting the position of the 'P' clip for the microswitch wires) and pull the beam out. If you want, remove the microswitch screws to completely free the beam from the printer.

The printhead is carried on a pair of arms with adjusting cams at the lower ends. Remove the self-locking nuts from the fixing screws and remove the springs, cams and screws from the printer. At the rear edge of the **Printer**

Driver PCB, unplug the edge connector that carries the motor and switch wiring and then remove the PCB and printhead from the chassis.

To separate the printhead from the **Printer Driver** PCB, remove the setscrews in the clamp. Slide the printhead, the flexible PCB tail and the clamp bar out of the block. Handle the printhead with great care, it has a brittle ceramic substrate and the only source of replacements is another HP9810 or HP9820 printer.

Back on the chassis, remove the circlip on the platen (upper) spindle at the front left side and slide off the gear. Loosen the 4 motor mounting screws and slide the motor forwards to relieve the belt tension. Remove the sprocket from the loading belt (lower) shaft at the front left, it is supposed to turn freely on this shaft, and remove the drive belt. On the right hand side, remove the circlips and gears from both shafts, the larger gear fits on the loading belt (lower) shaft.

On the underside of the printer, you will see the 2 loading belts that feed the paper towards the platen. Remove the circlip from one end of the rear belt shaft and slide this out, recovering the plastic rollers. Moving towards the front of the chassis, remove the (smaller) circlip from one end of the next shaft and slide that out too, again recovering the rollers. This shaft also carries the paper-out contacts, which can be left hanging on the wires.

There are actually 2 types of paper-out contact. One type has one wire connected to this contact assembly and the other soldered to a metal plate on the front section of the printer chassis. The other has both wires connected to separate contacts on the assembly carried on this spindle, they are connected together by coming into contact with an isolated metal plate on the front section of the chassis when the paper runs out. This makes little difference to the dismantling and repair procedures though.

The platen and loading belt shafts are carried in bearings in plates screwed to the sides of the chassis. Remove this assembly, note the positions of the plastic spacer strips under them. There is a thin idler roller carried in holes in these palates that may fall out at this point, make sure you don't loose it.

Now, on the right side of this assembly is the red manual paper freed leaver. Turn this through 180° to free it from the left bearing plate (this will take some force if the platen has not decayed) and then separate the various parts.

On the main printer chassis again, remove the 2 rear through-bolts. Push the rear chassis section downwards and unclip the paper guide wire. Remove the 2 front through-bolts, the top one carries the threaded spacer for the **Printer Interface** PCB, and separate the chassis parts. Remove the 4 screws loosed earlier and separate the motor from the left side plate.

Clean all the parts of the printer. If the platen has decayed, it will take

a lot of elbow-grease to remove the goo from the various chassis parts,m the **Printer Driver** PCB and the printhead. But clean the latter with care, it is very easy to damage

13.1.2 Dismantling the motor

Once the motor has been removed from the printer it can be dismantled to inspect and lubricate the bearings or repair winding faults.

Check the windings for continuity between the appropriate pins of the edge connector that plugs into the **Printer Driver** PCB. I have never had to do it, but it should be possible to re-wind the motor once it is dismantled.

Remove the rear end cover by unscrewing the 3 screws on it. These screws tap into pillars which are fitted with plastic sleeves, remove these sleeves next. Loosen (or remove) the 2 setscrews in the motor sprocket and slide this off the spindle followed by the plastic bush under it. Remove the circlip and the shim washers under it and slide the rotor, with more shim washers on the shaft, out of the rear end for the motor. Make sure that no shims have been left adhering to the bearing faces, or they are likely to be lost. Treat the rotor with care, it is fairly brittle and spares are difficult to obtain.

To dismantle the stator, use a suitable bar (such as the shaft of a large screwdriver) between the pillars that take the rear end cap screws to rotate the locking plate and free it from the motor casing. Remove the plate and the large spring washer under it, then lift out the stator pole pieces and the coils, keeping them in order. Ensure you refit the coils the same way up, if one (and only one) is turned over, the motor will run in reverse when it is reassembled. This causes no damage but you will have to take everything apart again to turn a coil over. The metal band around each coil slides off followed by plastic covers. At this stage it is possible to re-wind the motor.

Reassemble the motor is the reverse order to dismantling it and lubricate the needle-roller bearings with high melting point grease.

Next, reassemble the main chassis. If the platen is in good condition, reassemble the platen, bearing plates and belts too. But do not refit the PCBs or print head just yet. If the platen has decayed, the next section will tell you how to repair it.

13.1.3 Repairing the platen roller

To repair the platen the best material[1] appears to be 3M 8242-8 Cold Shrink(TM) kit[2]. This kit contains a length of the Cold Shrink tubing on a plastic support, 2 sleeves, and a packet of silicone grease. Ignore the grease, you do not want the platen to slip once assembled.

Push the 2 sleeves onto the ends of the original platen spindle (after cleaning the latter, of course), so that they meet near the middle of the spindle. It will take considerable force to do this. Make sure they go on evenly and touch, but don't force them on so far that there is a bulge at the joint between them

Slide the Cold Shrink over this and start to pull out the helically-wound support. When about half the support is removed, cut round the Cold Shrink itself and reposition it over the layer already fitted. Remove the remainder of the support.

Thus the spindle is covered with the sleeves and 2 layers of Cold Shrink tubing. The next job it to cut it to length. Rest the roller on the partially-open jaws of a bench vice and use a sharp knife to trim off the ends. Make the platen too long at both ends at this stage.

Clamp the spindle in the 3-jaw chuck of a small lathe and support the other end on a tailstock centre (HP kindly provides centring holes at each end of the spindle). Hold the knife against the toolpost and position the carriage so that the knife will cut that end of the platen at the correct place. Then while *turning the lathe by hand* (Don't even thin of running the lathe under power unless you want to make a trip to hospital – or worse – when the knife ends up in you), cut the sleeving to length. Reverse the platen in the lathe and cut the other end.

You should now have a repaired platen of the correct size. Reassemble the platen, belts and bearings and refit this assembly to the front of the printer chassis. Reassemble the gearing.

13.1.4 Checking and refitting the print head

the printhead has 88 connections in a row along the bottom edge which connect to corresponding pads on the flexible PCB that's integral with the **Printer Driver** PCB. These pads, starting at the left side of the printhead follow the pattern of 10 individual resistor connections followed by the common for those 10 resistors, followed by another 10 individual connections,

[1] Suggested by David Smith and Katie Wasserman on the Museum of HP Calculators Forum

[2] Available from RS Components, 251-7566 or Farnell 126-6864

their common and so on for a total of 8 groups.

Begin by testing the printhead. Check the resistance from each individual connection to the corresponding common one. A new printhead would have resistors measuring between 120Ω and 260Ω. The resistance may increase with wear, but it should still be less than 300Ω. If any resistors are totally open-circuit, then I am afraid the printhead needs replacement. The only source is another machine.

If the printhead is good, fit the clamp bar to the clamp block and position the head and flexiprint so that the contacts are aligned. Carefully tighten the setscrews. Do not overtighten them, there is a very real risk of cracking (and ruining) the printhead.

Now check that the printhead is connected properly by measuring the resistance between the collector (or metal case) of the transistor Q4 on the **Printer Driver** PCB and the cathodes (banded end) of each of the diodes in the row alongside it. Then check from the collector of Q3 to the cathodes of the next row of diodes, the collector of Q2 to the next row and finally the collector of Q1 to the cathodes of the front row of diodes. In each case you should see the resistance of a printhead resistor.

If some or all are open-circuit, release the clamp, reposition the head and flexiprint and try again. I am sure there was a jig at the factory for this, but it's not that difficult to do by hand.

When this is correct, refit the remaining mechanical parts and the PCBs to the printer chassis. Then fit the printer into the machine (it is not necessary to fit the fixing screws, just plug the **Printer Interface** PCB into the backplane) ready for electronic troubleshooting.

13.2 Printer electronic troubleshooting

With the printer fitted and the machine turned on. press the manual paper feed lever at the right hand side of the printer. The printer motor should turn.

If it doesn't, begin by checking **MotorClk**, for example at the output of U96 on the **Printer Interface** PCB. If it is not pulsing, check that the manual feed clock astable is running (there should be a square wave at the collector of Q21) and that **Feed/** at the collector of Q24 is being asserted. Then work through the few logic gates involved.

When **MotorClk** is behaving correctly, check that **MotorEn/** at the output of U5b is being asserted (low). Next, on the **Printer Driver** PCB check that the motor state counter is running. If so, then check the motor driver stages.

Once the printer will feed paper correctly, insert a roll of the correct thermal paper and try to print something. There are a number of possibilities:

The printer prints correctly : Not a lot more to be said, all you have to do is screw the printer down and it's done.

The paper feeds but nothing appears on it : First check the paper is in the right way round with the thermally-sensitive surface against the printhead. The method I use for this is to hold the end of the paper over a hot soldering iron. It will soon be apparent which side changes colour. Then try adjusting the printhead height using the adjustment cams so that the row of heater elements is against the paper. It may be necessary to remove the head and reposition it in the clamp. Check that **PrtHdEn/** is being asserted and that U4 on the **Printer Driver** PCB is enabling the printhead selection transistors,. It is possible, albeit unlikely that all the drivers have failed at once, it may be worth checking this if not other fault can be found

It prints, but some dots are missing : If only a few odd dot-columns are blank, then it is likely the fault is either a defective printhead or the appropriate diodes on the **Printer Driver** PCB. If an entire electrical row or column of the printhead matrix is missing, then investigate the driver for that line.

Nothing happens at all : First check that the printer doesn't think it's out of paper. The collector of Q23 on the **Printer Interface** PCB should be high. Then check if **PrtEn** is being asserted. If not, check the logic around the SR flip–flop U4b and U12a on the **I/O Interface** PCB. If **PrtEn** is being asserted, check that it is asserting **PrtHdEn**via U7a on the **Printer Interface** PCB, and check the motor signals as above.

Once the printer is correctly working, refit the 4 screws that anchor it to the main chassis.

Chapter 14

Repairing the HP9830 Cassette Drive

14.1 Overhauling the tape drive mechanism

The HP9830 tape driver suffers many fewer mechanical problems than the card reader or printer of the HP9810 and HP9820 machines,and that that do occur are generally easy to repair. It is necessary to remove the mounting bracket and front panel for almost all repairs, the head plate and drive spindles can then be dismantled as necessary.

14.1.1 Initial dismantling

Begin by removing the rear mounting bracket which is fixed by 4 screws. Unplug the 4 'plugs' (actually PCBs with pins soldered to them) from the **Tape Drive Connector** PCB.

These plugs connect to the following parts of the drive :

S – Solenoid (9 pins) : The 2 solenoids that rock the motor beam and the 'write protect' and 'cassette in' microswitches

M –Motor (5 pins) : The 2 drive motors

P - Photo (5 pins) : The optical EOT sensor

H- Head (9 pins) : The **Tape Head** PCB

The **Tape Drive Connector** PCB is held to the bracket by 3 screws, loosen these a few turns at a time to separate the PCB from the bracket. Normally the screws and spacers stay with the bracket, the threaded ends of the spacers unscrew from the tapped bushes on the PCB.

Next remove the tape drive front panel. It is retained by a long through-bolt. With the tape drive front-upwards. remove the nut, washers and spacer from the end of this and then pull out the bolt with a further spacer and washers. You may want to store these parts on the bolt and refit the nut by a few turns. Lift up the front panel and slide out the window. Remove the nut and bolt holding the earth wire to the front panel and remove the panel entirely.

The motors are removed next. Take out the screw on the motor beam pivot post and the 4 screws holding the rear bracket to the sides of the drive. Take off this bracket and slide the motor assembly off the pivot.

14.1.2 Dismantling the head plate

The tape head is connected to the **Tape Head** PCB by 4 wired fitted with push-on contacts that go onto the pins of the head. Remove these, noting their positions then remove the 2 screws and spacers holding the PCB to the head plate assembly. Put the PCB aside.

Remove the 3 screws holding the left-hand cassette guide to the head plate Ease off the guide and unhook the cassette holding leaf spring from inside the frame. Remove the 2 bushes form their pins on the chassis (these often fall off if you are not careful). Free the right hand guide from the similar bushes on the right side of the drive chassis and remove the complete head plate. Again remove the bushes before they fall off and get lost.

The optical EOT sensor PCB is retained by 2 further screws. Remove the guide from the front of the sensor and remove the PCB complete with the LDR and light bulb. This bulb is one of the common failures of the tape drive, it is a 5V 60mA T1 wire-ended bulb[1] You can easily check this for continuity and solder in a replacement at this stage.

It is not advisable to remove the head, it would have to be realigned when it is refitted. I have never done this, but maybe one day...

14.1.3 Dismantling the drive spindles

On the main tape drive chassis, remove the motor beam pivot post which is retained by 1 screws.

It is possible to remove the drive spindles without removing their leaf springs first, but it is a little tricky. It is a simpler to remove the single screw from the rear face of the drive wheel and lift off the leaf spring. Then remove

[1]For example Farnell 113-9296

the circlip and plastic washer from the front end of the spindle. slide the spindle out rearwards, and recover the second plastic washer.

The 'tyre' on the drive wheel is the other main failure point of this drive. It is a size BS219 O-ring[2] and can easily be replaced at this point by stretching it over the drive wheel.

The brake plate is retained by 2 screws. Remove these, free the return spring and remove the plate and its spacers. The latter are asymmetrical and must be refitted the correct way up.

14.1.4 Initial electrical tests

Clean all the parts of the tape drive, paying particular attention to the head face.

Some of the electrical parts of the tape drive should be tested at this point. The resistance of each drive solenoid (measured between the pins on the connector 'plug' PCB should be around 22Ω. The drive motors, again checked at the pins on the 'plug' should be around 10Ω. Also check the filament lamp in the EOT sensor if you have not already done so. The head windings can be checked for continuity, but as in the case of the card reader use a low-current tester and demagnetise the head afterwards.

Then reassemble the tape drive and fit it, and the 4 PCBs in the tape control section, to the machine.

14.2 Repairing the tape drive electronics

Although the tape drive controller is much more complicated than, say, the card reader system of the HP9810 or HP9820, it is quite straightforward to repair. One reason is that HP intended it to be repaired to component level in the field, and provided some very useful testpoints on the PCBs. The HP service manual for the HP9830 includes some information on troubleshooting this section, but it suggests you find the faulty board by boardswapping, which is something that I will never recommend.

Begin by pressing the rewind key on the keyboard. This will set U4a on the **Tape Controller** PCB (even if the reset input of this flip-flop is low, the Q output will be high while the rewind key is pressed). This will cause all 3 control signals to the **Motor Control** PCB to go low, The appropriate solenoid and motor should operate turning the supply side spindle. If not, then check that TP5, TP7 and TP8 on the **Tape Controller** PCB are all low. If not, troubleshoot the the motor control logic on the **Tape Controller**

[2]This O-ring is 1+9/16" outside diameter, 1+5/16" inside diameter and 1/8" thick

PCB. If these signal are all low, check the direction control logic on the **Motor Control** PCB to fund out why the solenoid isn't engaging or the speed control circuit on the same PCB id the solenoid operates but the motor doesn't.

Now take a scratch tape, and using an audio cassette recorder, fast-forward it off the clear leader. Put this in the HP9830 drive and press the rewind key again. The rewind flip-flop should now latch on, the tape should rewind to the leader and stop. Problems here can be caused by the rewind flip-flop, U4 on the **Tape Controller** PCB, the end-of-tape logic on the **Tape Interface** PCB, or the EOT sensor itself. It is also worth checking that **CassSw** is high, If the microswitch in the tape drive is not properly detecting the cassette, U4a's reset input will be kept low.

Next initialise the tape in the usual way. The forward drive should now engage (on the **Tape Controller** PCB, TP5 should be low and TP7 and TP8 high). If these signals are incorrect, check if **TapeSel** (on TP2 on the **Tape Controller** PCB) is pulsing high. If not, check the selection logic on the **Tape Interface** PCB and if necessary the **CEO** signal logic on the **I/O Interface** PCB. If **TapeSel** is pulsing high, check U10, U8 an the **Tape Controller** PCB.

If the motor control signals are correct but the tape motion isn't, again investigate the **Motor Control** PCB as described above.

While the tape is initialising, the write signals should all be active. Check **WrClk** on TP10 of the **Tape Controller** PCB, write data on TP9 and **WrMark** on TP10. If they are all missing, check that the write clock oscillator is running (TP1 is the output of this circuit). Also check that the bit counter and the data shift register are working correctly. Moving to the **Read/Write** PCB, the BMS-encoded data can be checked on TP2 and TP3. Unfortunately the 5V supply to the write buffer on the **Tape Head** PCB is not available on a test point but it may be worth checking that this is present if you have problems with writing.

If you get an error suggesting that the tape is write-protected, the first thing to check is the write protect microswitch on the tape drive itself. Mechanical switches cause a lot more problems than electronics**!**

If you have a printer, next try listing the tape. This will check the read function. During this you should see transitions on TP6 and TP7 of the **Read/Write** PCB (if either is missing, check the read amplifiers and the head itself) and then check there is **RdClk** on TP10, **RdData** on TP4 and **RdMark** on TP9. If these are correct but the listing operation fails, check the data shift register and associated control logic on the **Tape Controller** PCB and the flag logic on the **Tape Interface** PCB.

If you don't have a printer, try saving a short program to the tape. At

the start of the save operation, the system reads the file headers from the tape and thus the read signals can be checked during this time. Then try loading the program back in, if this works, the tape system can be considered to be operational, if not, the fault is probably on the **Tape Controller** PCB around the data shift register or the **Tape Interface** PCB around the data flag logic.

14.3 Beeper

The beeper is simple to troubleshoot. Perform some operation that causes an error, and which therefore should cause the beeper to sound. If it does not, check the **CROut** signal from the **I/O Interface** PCB is being asserted, if not, check the source of this signal. If it is, check U12 on the **Tape Interface PCB** is changing state.

Then check that the oscillator U15a and U15b is running, and that during the beep a square wave appears on the output of U15d. Finally check the loudspeaker driver transistors and their associated components.

Chapter 15

Final Comments

Now all the electronic and mechanical sections of your HP9800 calculator have been repaired, all that remains is to complete the reassembly.

In an HP9830, refit the **Display** PCB fixing screw and check that the tape drive and expansion ROM unit are screwed in place. In an HP9810 or HP9820, check that the card reader and printer are screwed down. Refit the keyboard, hooking the bezel behind the printer paper guide if appropriate, and fit the 4 fixing screws on the underside of the machine, starting with the 2 nearest the back which screw into the metal pillars on the keyboard.

Do one final check of the machine's operation before refitting the PCB hold-down bars in an HP9830 and the top cover.