

Epson HX-20 Tips and Tricks

Martin Hepperle, November 2018 – January 2024



Contents

1. General	2
2. Power Supply	3
2.1. Transformer Unit	3
2.2. Replacing the Battery	4
2.3. Charging the Battery	5
3. Variations of the ROMs	5
4. New Printer Paper	5
5. New Printer Ribbons	6
6. Internal RAM Boards	7
6.1. “mc” 8 KB RAM board	7
6.2. 16 KB RAM board Type 1	8
6.3. 16 KB RAM board Type 2	9
7. HX-20 for the Bundeswehr	10
8. Replacing the Capacitors	10
9. Replacing the Cassette Drive Belt	11
10. Character Sets and Keyboards	14
11. Keyboards Types	15
12. Loading BASIC Programs via RS-232C	17
13. Controlling External Devices	17
14. Some Useful Subroutines	18
14.1. User Defined Characters	18

14.2. Get the Time in Seconds	18
14.3. Functions to obtain Low and High Byte of an Integer	19
14.4. Decoding a String with a Hexadecimal Number	19
15. Some Benchmark Results	19
16. Writing Machine Language Routines	21
16.1. Extending the Operating System	27
16.2. Some Details about HX-20 BASIC (Microsoft BASIC)	34
16.2.1. The Floating Point Accumulator	34
16.2.2. Memory allocation of Arrays	34
16.2.3. The BASIC Work Areas	35
17. Using a Printer	37
18. MH-20 – A Peripheral Emulator	37
18.1. Required Hardware for HX-20	38
18.2. Using the MH-20 Software	38
18.3. Display Controller Emulation	39
18.3.1. Applicable BASIC Keywords and Commands	40
18.4. Disk Drive Emulation	44
18.4.1. Technical Background	44
18.4.2. The Emulation	45
18.4.3. Applicable BASIC Keywords and Commands	45
18.5. Credits	46
19. Map of the System RAM	47
20. News and Commercial Announcements	49
21. References and Further Reading	59

1. General

The HX-20 was, and still is, a handy, portable computer with built-in printer and cassette drive – some call it the first laptop.

The LCD screen shows a window of 4 lines of 20 characters each into a virtual screen which can (in theory) be as large as 255 by 255 characters. In addition to text it can also display graphics at its resolution of 120x32 pixels.

The cassette drive can be replaced by a small ROM box and you can add a larger RAM/ROM box to the left side of the computer and you can install one ROM-chip inside the computer.

Additional devices like a barcode reader, a flexible disk drive unit and a display controller were available in those days.

The operating system and an adapted Microsoft BASIC are stored in 32 KB of ROM, which also contains a Monitor program. Furthermore 16 KB of RAM are installed inside the computer. The BASIC also provides commands for graphics and for the RS-232C interface. It can also call routines in machine code. Programs and data files can be stored in RAM and are immediately available after switching the device on.

The serial RS-232C interface can be used to communicate with other computers or printers and modems. A second „high speed interface“ was intended to be used by disk drives and display controllers. It is not directly supported in BASIC, but can be used by programs in machine language.

The HX-20 computer was often used by sales forces, in surveying, agriculture and for mobile data acquisition or even by the military. For these applications additional peripherals have been constructed and can sometimes be found installed on these systems.

Because of the robust mechanical design the HX-20 is a long lasting computer – except for some aging problems of its electronics components.

2. Power Supply

2.1. Transformer Unit

The transformer unit for the HX-20 should never be used without the built-in battery. On the one hand side the battery acts as a buffer for actions with high power demands, for example printing or accessing the cassette drive. Peak currents can exceed 1 A. On the other hand the battery charging load reduces the voltage of the transformer to the required voltage of about 5 V.

The charging time of the original Ni-Cd cells (having about 1100 mAh) is roughly 8 hours. When new cells with a higher capacity of 2000 mAh are used, the charging time grows to 14 hours. In order to maximize battery life you should avoid overcharging the battery.

The original transformer unit is matched to the battery circuit of the HX-20. It supplies its nominal voltage of 6V at 600 mA only when it is loaded by charging the battery. The 5.5/2.1 mm barrel plug carries plus on the outer barrel and minus on the inner pin – most standard power supplies have the polarity reversed. The circuit in the HX-20 has a protection diode so that no damage can occur when the polarity is incorrect, but also no charging will take place.

You should always discharge the battery until the „CHARGE BATTERY !“ message appears, perform a full charge and then disconnect the power supply again.



Figure 1: The original power supply unit says “6 V” on the label.

Measurements show that the original power supply delivers about 9 V when unloaded, which results in an initial charging current of 250 mA. During charging the current drops rapidly down to 150 mA. When the battery voltage has reached its level of about 6V, the current has fallen to about 50 mA.

A modern regulated power supply of 6 V produced a low initial current of only 50 mA which quickly drops to 20 mA. After about two hours the current has become zero and the battery will never be fully charged.

Therefore, a replacement power supply must deliver about 9V and the charging current must be adjusted by inserting a suitable resistor into the cable. The average current should reach about 1/10 of the battery capacity (i.e. 200 mA for a 2000 mAh battery).

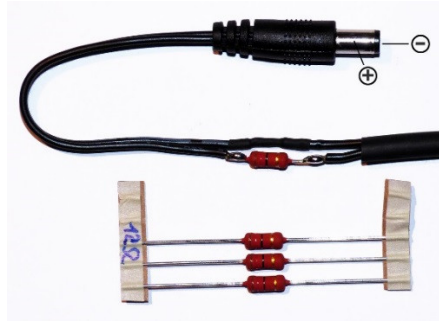


Figure 2: Using a modern, stabilized 9V/4.5W power supply with an inline 2 Watt resistor of 12 Ω yields an average charging current of 200 mA and a charging time of about 12-14 hours. The cable has to be cut anyway to reverse the polarity. Do not forget to slide the shrink tubing over the cable ends before soldering.

2.2. Replacing the Battery

- Ready-made battery packs with connectors can be found on eBay. I cannot say anything about their quality, but I would guess that they work fine. If you have the equipment, I recommend to charge and discharge the battery at least once using an external charger/discharger to determine their true capacity. Alternatively you can build your own battery pack from single NiCd cells. NiCd chemistry is preferable because the simple charging circuit (a resistor and a protection diode) in the HX-20 is designed for these cells. The cells must not be too large – there are small differences between so called „Sub-C“ cells and it is better to use smaller cells than to try to maximize the capacity. A capacity of 1000-1600 mAh is sufficient – you do not need 2500 mAh.
- When working on the HX-20 you must avoid electrostatic charges. Use a grounded metallic or conducting foam work surface and ground yourself using a wrist strap.
- Place the computer with the keyboard facing down on a soft mat.
- Remove all seven screws on the bottom side and put aside.
- Turn the computer over, keeping the upper and lower shells together.
- Lift the upper shell at the rear end by about 5 cm. Use the front edge as a hinge. Next you can unlock the flexprint cable beside the battery pack by pulling the collar upwards. Pull the ribbon cable carefully upwards, out of the connector.
- Now you can open the case completely, again using the front edge as a hinge. Careful with the two ribbon cables close to the front edge. You can lay both halves flat on our working surface, keeping the two ribbon cables in their connectors.
- Remove the screw in the metal plate over the battery pack and unhook the plate from the case.
- Place the new battery close to the computer – if you replace the battery within a few minutes, memory content will be maintained.
- Pull the old battery out of the cavity and unplug the connector.
- Plug the new battery in and place it into its cavity.
- Insert the metal plate and tighten the screw lightly. In case of a home-made battery pack: be sure that you do not create a short – the energy content of the battery pack can lead to a fire.
- Use your left hand to hold and fold the upper case back over the lower case, using the lower edge again as a hinge. Hold the rear open and insert the flexprint cable and close the lock by pushing the collar down, all with your right hand.
- When the case is completely closed, wiggle the lever under the microcassette drive (or ROM box) slightly right/left to make sure it locks into its counterpart.

- Also make sure that the blue cloth ribbon in the printer bay is properly placed and not caught between the case parts. Also check the proper routing of the printer paper.
- Check the proper placement of the panel with the serial connector cutouts in the rear wall.
- Before replacing the screws: test the system – if you obtain no display you might have to reattach the flexprint cable properly.
- If everything works: replace the screws and pull then hand tight.

2.3. Charging the Battery

The battery should only be recharged when the HX-20 tells you to do so. After charging, the charger should be unplugged. Figure 3 shows a time history of the charging current obtained with a 9 V power supply and a series resistor of 12 Ω . The charging was initiated after the HX-20 signaled “CHARGE BATTERY !” and a minimum of the current indicates the completion of the charge. A charging time of about 12 ± 1 hours seems to be adequate for the 2000 mAh cells and this charger.

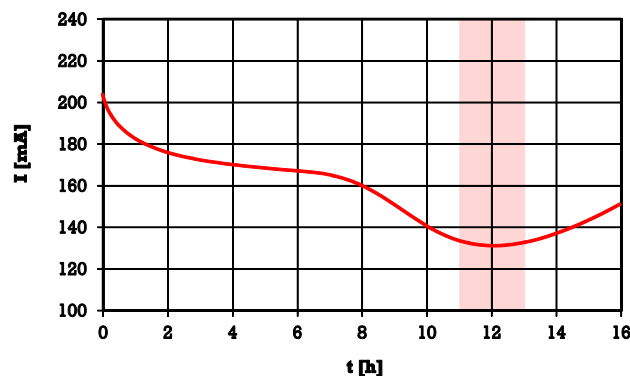


Figure 3: Charging current versus time for a NiCd battery pack having a nominal capacity of 2000 mAh.

3. Variations of the ROMs

In Europe, there are at least two versions of the ROMs: they boot as BASIC V1.0 and BASIC V1.1.

The HX-20 cases also differ slightly: older ones have an opening in the bottom cover where the auxiliary processor is installed, while the later ones do not have this additional opening. So far I encountered four systems:

- SN 011359, BASIC V1.0: has opening over slave processor
- SN 020734, BASIC V1.1: has opening over slave processor
- SN 040576, BASIC V1.1: has no opening over slave processor
- SN 042951, BASIC V1.1: has no opening over slave processor

4. New Printer Paper

- You can use any non-thermal printer paper with a width of 57...58 mm. In order to fit the tight space you probably have to roll-your-own from a larger roll of paper. Just take a pencil and wind a few meters of paper around it, keeping its side edges neatly aligned, remove the pen and you are ready to go.

5. New Printer Ribbons

- In most cases the old ribbons are dry and produce only weak printout if any. Also the foam rollers are disintegrating after so many years. Therefore, they tend to block the motion of the endless ribbon. Luckily, even in 2018 new cassettes are still available, because they seem to be used in printers of some Point-Of-Sales systems.

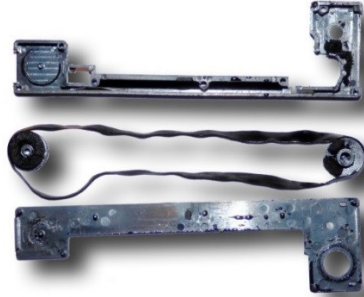


Figure 4: This ribbon cassette was taken apart to show the internal structure and the disintegrating foam wheels.

6. Internal RAM Boards

Some HX-20 come with an internal memory expansion. Originally Epson had not planned to allow for internal RAM extensions, but some tinkerers found out, that there was enough space inside the shell to add a board between keyboard and motherboard. A connector could be clamped onto the solder side pins of the external bus connector at the left edge of the case. This connection is the weak point of all boards – malfunctions are usually resulting from poor contact and I had to replace the flat spring connectors with strips from a “tuned precision socket” on the “mc” board to make it work again.

The issue April 1984 of the German computer magazine „mc“ (“MicroComputer”) presented a do-it yourself circuit layout for an 8 KB RAM expansion board. If no ROM modules were used, two of these „mc“ boards could be added for the maximum of 16 KB RAM.

Similar boards were also produced by various manufacturers. These commercial boards usually came with 16 KB of RAM or ROM, which could be selected by a setup procedure with the monitor.

6.1. “mc” 8 KB RAM board

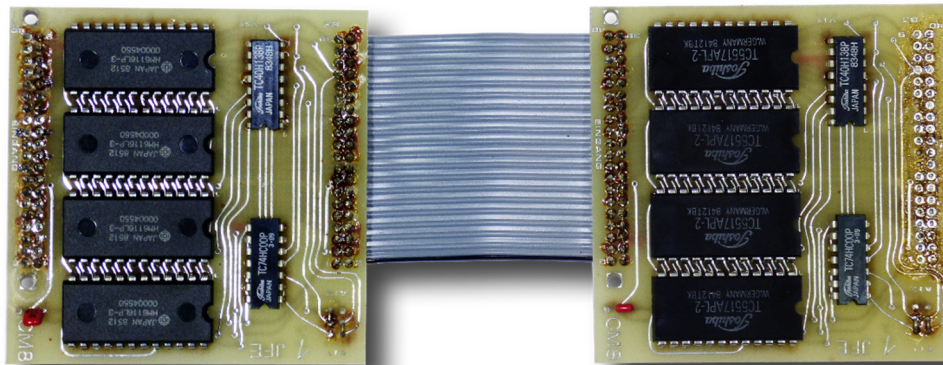


Figure 5: A set of two RAM boards as published in “mc” magazine. Both modules are identical and can be switched to a starting address by a solder bridge (a dip switch in the published design). Another switch can be used to deactivate each board if a ROM would be installed.

These boards require no special activation. One or two boards can be installed inside the HX-20, adding 8 to 16 KB of RAM. After installation, the usual full reset sequence is applied:

- Reset (press Reset button)
- Initialise (CTRL+SHIFT+@) / (CTRL+SHIFT+§)
- Start BASIC (2)
- Input `PRINT FRE(0)` (Return)

The result should be 29275.

6.2. 16 KB RAM board Type 1

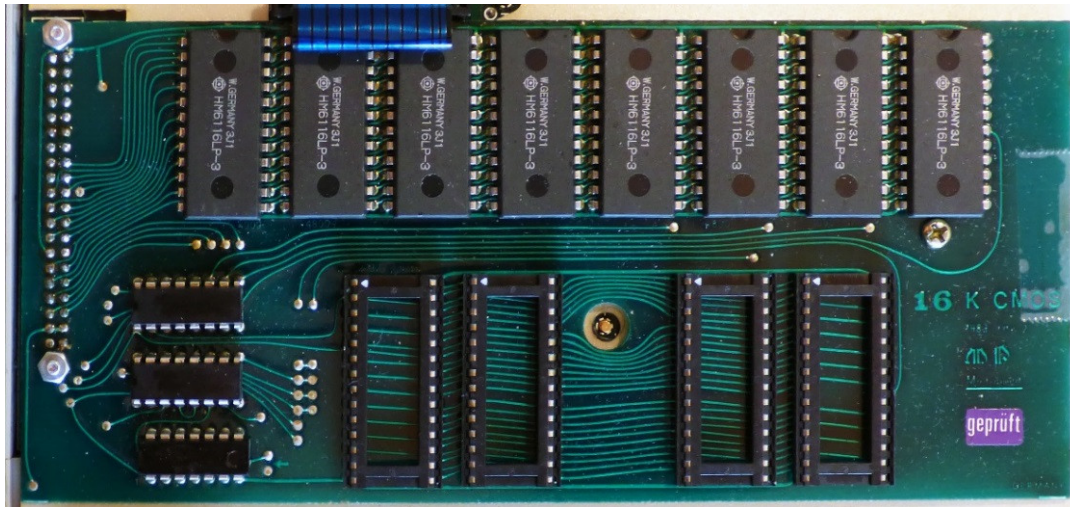


Figure 6: RAM board Type 1 with eight 2 KB RAM chips and four additional ROM sockets.

In order to make the full RAM capacity available the following procedure has to be applied:

- Reset (press Reset button)
- Initialize (CTRL+SHIFT+@) / (CTRL+SHIFT+\$)
- Start Monitor (1)
- Input S7E (Return) [setting \$7E to \$80 allows
- Input 80 (Return) accessing I/O address \$3B below]
- Input - (Return)
- Input S3B (Return) [setting \$3B (undocumented)
- Input 82 (Return) to \$82 obviously enables RAM]
- Input - (Return)
- Input B (Return)
- Initialize (CTRL+SHIFT+@) / (CTRL+SHIFT+\$)
- Start BASIC (2)
- Input PRINT FRE(0) (Return)

Again, the result should be 29275.

6.3. 16 KB RAM board Type 2

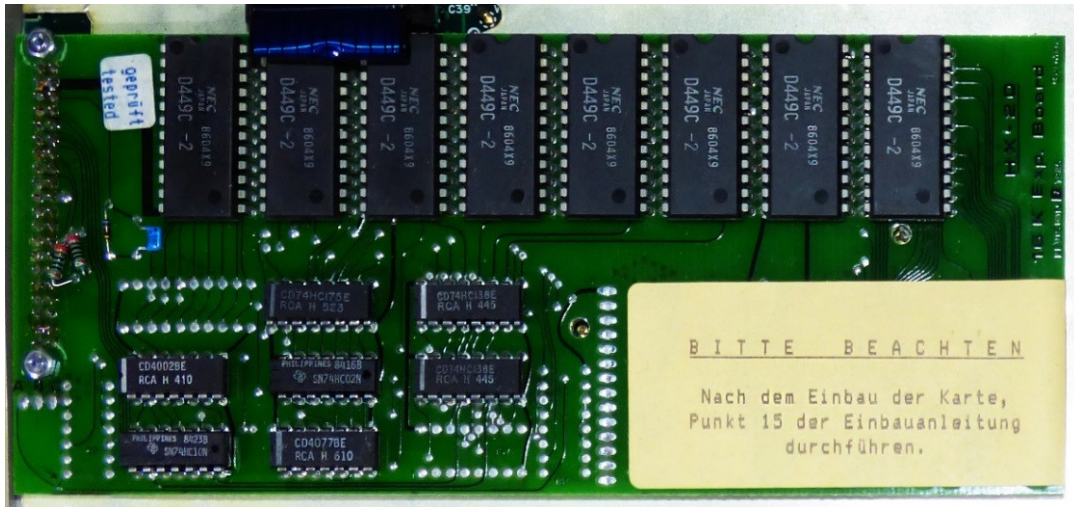


Figure 7: RAM board Type 2 produced by Steinwald with eight 2 KB RAM chips (the empty footprints under the sticker and the supporting TTL chips can also be populated with bank switched RAM).

The activation sequence for accessing the full 32 KB RAM for this board is:

- Reset (press Reset button)
- Initialize (CTRL+SHIFT+@) / (CTRL+SHIFT+\$)
- Start Monitor (1)
- Input **SFFF5** (Return) [setting this byte in high address range]
- Input **0** (Return) enables 32 KB RAM]
- Input **-** (Return)
- Input **B** (Return)
- Initialize (CTRL+SHIFT+@) / (CTRL+SHIFT+\$)
- Start BASIC (2)
- Input **PRINT FRE(0)** (Return)

As above, the result should be **29275**. The option ROM socket on the main board cannot be used.

The activation sequence for accessing 24 KB RAM plus the 8KB ROM socket on the mainboard is:

- Reset (press Reset button)
- Initialize (CTRL+SHIFT+@) / (CTRL+SHIFT+\$)
- Start Monitor (1)
- Input **SFFF5** (Return) [setting this byte in high address range]
- Input **8** (Return) enables 24 KB RAM and 8KB ROM]
- Input **-** (Return)
- Input **B** (Return)
- Initialize (CTRL+SHIFT+@) / (CTRL+SHIFT+\$)
- Start BASIC (2)
- Input **PRINT FRE(0)** (Return)

The result should be **21083**. The option ROM socket can be used e.g. for a FORTH ROM. The remaining 8 KB of RAM on the board are “wasted”.

Note: you can also use **STAT ALL** to examine the memory configuration.

7. HX-20 for the Bundeswehr

The German Army used the HX-20 to determine firing tables for howitzers. Devices from old military stock appear regularly on eBay Germany, albeit at high asking prices around 100€ because these are offered by commercial dealers and gold diggers. Keep in mind that these devices have been modified and usually are not overhauled so that you will have to invest into a new battery as well as a replacement of the capacitors.

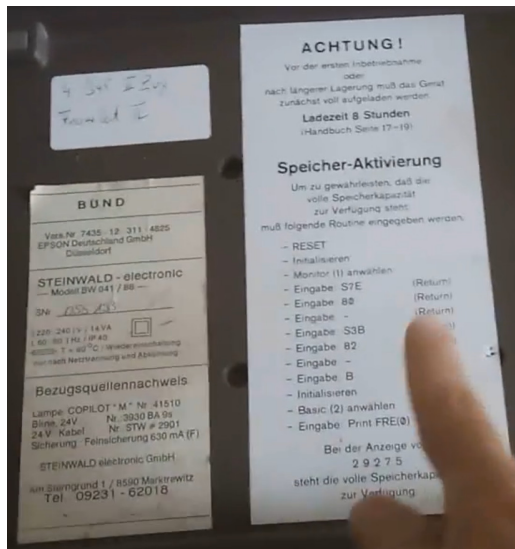


Figure 8: The instructions for activation.

These devices come in a modified suitcase with connectors for an external power supply and a reading lamp. They also have a memory expansion installed, which must be activated according to the instruction sheet.

The manufacturer of these modifications was:

Steinwald Electronic GmbH
Am Sterngrund 1
6590 Marktredwitz

Today the company name is:

STEINWALD Datentechnik GmbH
Oskar-Loew-Str. 12
95615 Marktredwitz



Figure 9: Some HX-20 come with a nice label template for tape operation.

8. Replacing the Capacitors

The HX-20 contains 14 electrolytic capacitors on its main board. These have exceeded their useful lifespan after more than 30 years. In most cases at least some are already leaking and the electrolyte can be found on the printed circuit board and in the gray discolored solder joints. When trying to run the HX-20 a weak or flickering LCD screen which cannot be adjusted to full contrast (all pixels dark) is a sign of bad capacitors. Then it is time to replace all of them. Besides a broken battery pack this seems to be the second most common problem with the HX-20.

The replacement is simple but tedious because the holes are relatively small and the old solder is difficult to remove. This is partially caused by the reaction with the electrolyte which seems to change the properties of the old solder. Despite some experience gained by refurbishing three HX-20, it usually takes me about two hours to replace all capacitors.

If available all capacitors should be of miniature size – you should revert to the standard size with a height greater than 7.5 mm only if you cannot source the smaller ones. The standard height capacitors must be mounted flat on the circuit board in order to fit the board into the case. In this case you have to bend the wires by 90 degrees. On the other hand this has the advantage that you can solder from

both sides and better inspect the soldering joints. I found the miniature capacitors at Reichelt Elektronik in Germany, however not for all required capacities.

The following electrolytic capacitors are required:

C1, C2, C3, C4, C5, C6:	10 μ F/16 V	4.3 mm \varnothing \times 7.5 mm
C7, C8	33 μ F/16 V	6.5 mm \varnothing \times 7.5 mm
C9, C10, C11, C12	47 μ F/16 V	6.5 mm \varnothing \times 7.5 mm
C13	100 μ F/6.3V	6.5 mm \varnothing \times 7.5 mm
C14	1 μ F/16V	6.5 mm \varnothing \times 7.5 mm

A professional solder sucker of the pistol type is a good tool to remove the old solder, but in some cases additional mechanical rework might be necessary. Be careful not to damage the through-hole connections between upper and lower board layers.

If you discover electrolyte on the PCB or on the lower side of the old capacitor some cleaning of the board with water and alcohol should be performed to avoid corrosion.

Be sure that the new solder flows freely through the holes so that both sides of the PCB are wetted. Wiggling each wire slightly before removing the soldering iron helps the tin to flow through the narrow gap. To be sure that each solder joint is nice and without stresses I even reflow each joint after cutting the excess wires.



Figure 10: Some of the nasty culprits.

9. Replacing the Cassette Drive Belt

Most HX-20 are equipped with a micro cassette drive. It comes not as a surprise that the belt of this drive ages and finally breaks.

It can be replaced by a rubber belt with a square cross section of 0.8 \times 0.8 mm to 1 \times 1 mm and a circular inner diameter of about 50 mm. This corresponds to a width of approximately 80 mm when pressed into a flat shape ($2 \times 80 \approx \pi \times 50$). The belts I used had a diameter of 49 mm and a nominal cross section of 1 \times 1 mm. The cross section actually measured more like 1.2 \times 1.2 mm which worked fine, but is the upper limit.

You need pointed tweezers, a small Phillips head screwdriver, a de-soldering tool and a soldering iron.

The parts include a few tiny M 1.4 screws, washers and spacers, which should be saved in a small container to avoid losing them. It may be wise to take some photographs or to make some sketches during the disassembly.

In order to replace the belt one has to partially dismantle the drive:

- Remove the drive box from the HX-20 by pushing the lever on the rear of the HX-20.
- Remove two screws from the bottom and take the bottom shell off.
- Remove the three 3 small screws holding the metal frame in the upper shell. Two screws above and below the connector and one on the opposite side.
- Unscrew the fourth screw with its small brass spacer at the upper edge of the PCB which fixes the PCB and the motor carrier in the upper shell.

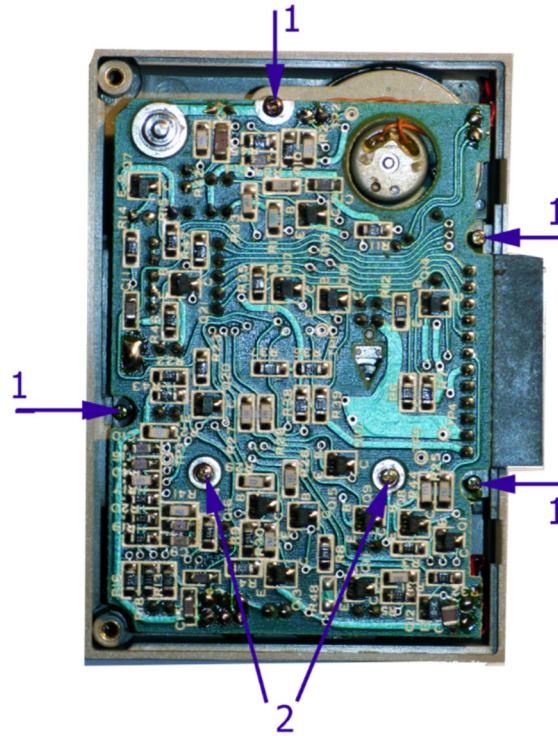


Figure 11: These screws have to be removed first:

- 1: Four screws to remove the drive assembly from the upper shell;**
- 2: two screws to remove the PCB from the drive frame.**

- Carefully remove the upper shell. Open the hatch and slide the shell off. There is a small internal sheet metal lever for pushing the hatch open. It can be rotated slightly around its vertical pivot axis to get out of the way. Do not use force, just wiggle the shell a bit and slide it off at an angle of about 45 degrees.
- Unscrew the two screws holding the PCB on the cast aluminum frame; take care of the two washers under the screw heads as well as the small stepped spacers under the PCB.
- Note the polarity and unsolder the two wires from the tachometer cap and both motor wires.
- Carefully unfold the PCB from the mechanical assembly. The remaining wires on one side serve as a “hinge”.
- Remove the metal bridge supporting the large drive wheel and the tension wheel (two screws).
- Unscrew the tachometer cap above the motor (2 screws plus 2 brass spacer tubes).
- Note: you might also want to inspect the three electrolytic capacitors. while you have the PCB on the table.

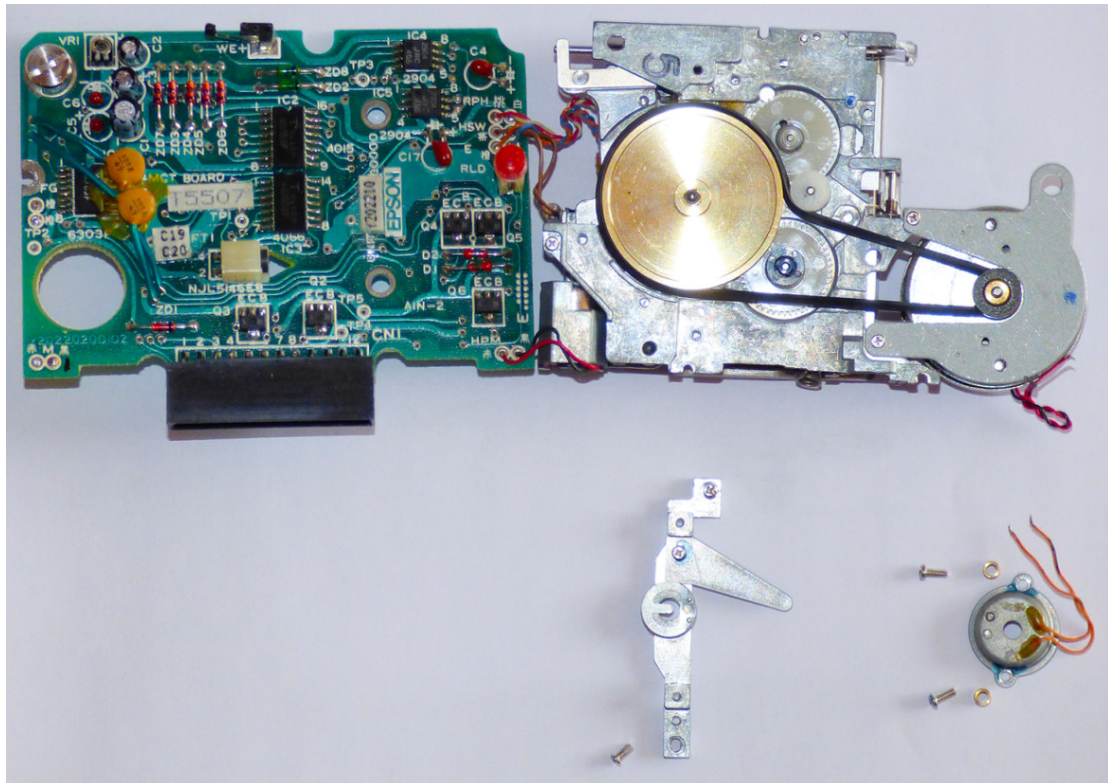


Figure 12: The PCB can be unfolded after unsoldering motor and tachometer cap wires and after removing the bar across the large drive wheel and the tachometer cap. The new belt has already been installed. You can also see three electrolytic capacitors on this board, which may also be replaced while the drive is already open.

- Remove the old belt; note how the small white wheel applies tension to the belt.
- Install the new belt – it should fit the grooves so that its cross section is angled at 45 degrees.
- Replace the mechanical parts.
- Turn the wheels manually to move the belt and make sure that it moves smoothly without rubbing against other parts.
- Replace all parts, except for the plastic shell covers.
- Solder the four wires back to where they belong.
- Plug the drive assembly into the HX-20 and make a test run (WIND, FILES, BREAK).
- If everything works, replace the two plastic shell parts.
- Make sure that the hatch can be opened with the lever; you may have to rotate the small internal sheet metal lever back so that it properly engages the hatch mechanism.

And that was it – phew!

10. Character Sets and Keyboards

The European ROM version of the HX-20 supports different character sets than the International or Japanese versions. For example the British pound sign is not present.

		country code							
		0	1	2	3	4	5	6	7
character code	35	#	#	#	#	#	#	#	#
	36	\$	\$	\$	\$	¤	\$	\$	¤
	64	©	©	©	©	©	§	©	©
	91	£	£	£	£	£	£	£	£
	92	\	\	\	ø	ö	ö	ö	ø
	93]]]	À	À	Ü	§	À
	94	^	^	^	Ü	Ü	^	^	Ü
	96	`	`	`	é	é	`	`	é
	123	€	€	€	æ	ä	ä	é	æ
	124	ı	ı	ı	ø	ö	ö	ü	ø
	125	›	›	›	ß	ß	ü	é	ß
	126	~	~	~	ü	ü	ß	ü	ü
country	SE		DE	FR	DK	SE	DE	FR	NO
	ASCII				national				

Figure 13: Character sets available in the European versions of the HX-20.

The country codes 0, 1 and, 2 have identical ASCII character sets, but different keyboard assignments.

These character bitmaps are stored in the last system ROM which is mapped into the memory range E000-FFFF. The following character bitmap patterns can be found at the given offsets into this ROM:

Offset 1BBE...1D9D: 96 characters of 5 bytes (shown below 6 bytes wide as they appear on screen)

```

! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

```

Offset 1D9E...1E5D: 32 graphics characters of 6 bytes

```

+ - = < > [ \ ] ^ _
~ a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

```

Offset 1E5E...1ED0: 23 international characters of 5 bytes (shown below 6 bytes wide as they appear on screen)

```

À Á Â Ã Ä Å Æ Ç È É Ê Ë

```

Figure 14: Character bitmaps in the system ROM of the HX-20.

Note that the given address ranges are for ROMs which show BASIC Version 1.1 on system start. The addresses in ROMs of Version 1.0 are shifted down by 8 bytes (the data starts at offset 1BB6). These addresses are valid for the European HX-20 models.

The character set can be switched by storing a byte between 0x10 and 0x17 (for country codes 0 to 7) at the address 0x7F and then executing the subroutine at 0xFF6A.

```

10 POKE &H7F,&H16
20 EXEC &HFF6A

```


11. Keyboards Types

So far I have encountered two different types of HX-20 keyboards. The first one uses individual mechanical key switches with flat spring contacts; the second type is built from two flexible membrane layers which carry conductive traces. Pressing a key deflects the upper membrane locally until it touches the lower layer. The first type is very robust and individual switches can be replaced or cleaned. The membrane-based keyboards seem to age less well and may develop problems depending on temperature or moisture. As the individual key mechanics are installed with melted plastic rivets, they cannot be removed without damaging them.



Figure 15: Top view of both keyboard types: the upper one is using membranes and assemblies of grouped switches, the lower one carries individually soldered in key switches.

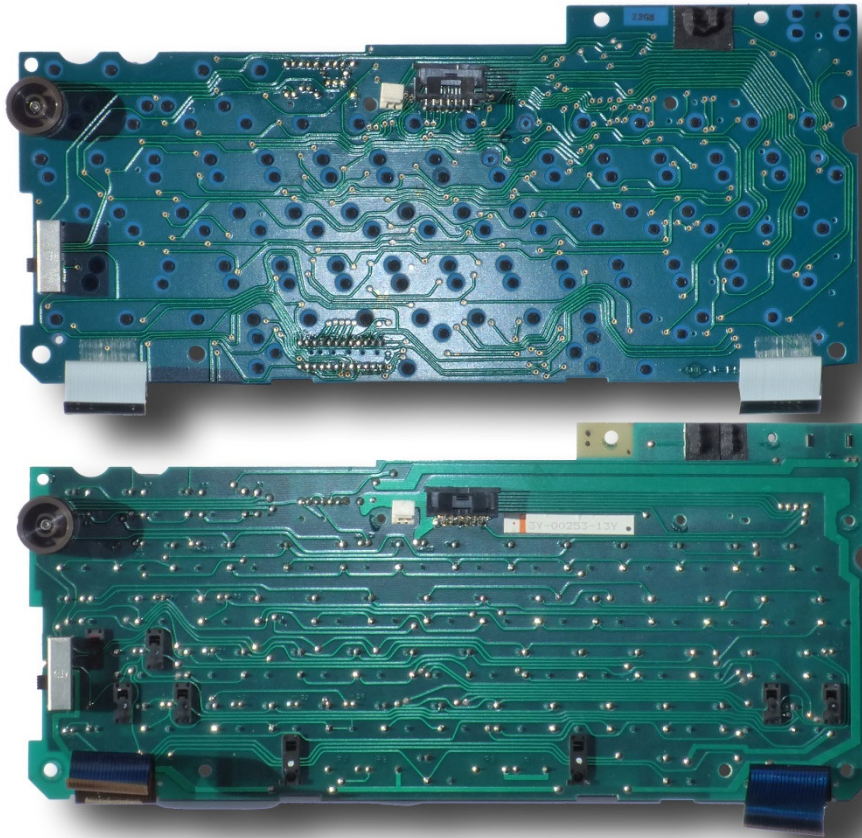


Figure 16: Bottom view: the upper keyboard shows the black melted plastic rivets of the switch frames, the lower shows the soldered in key switches,

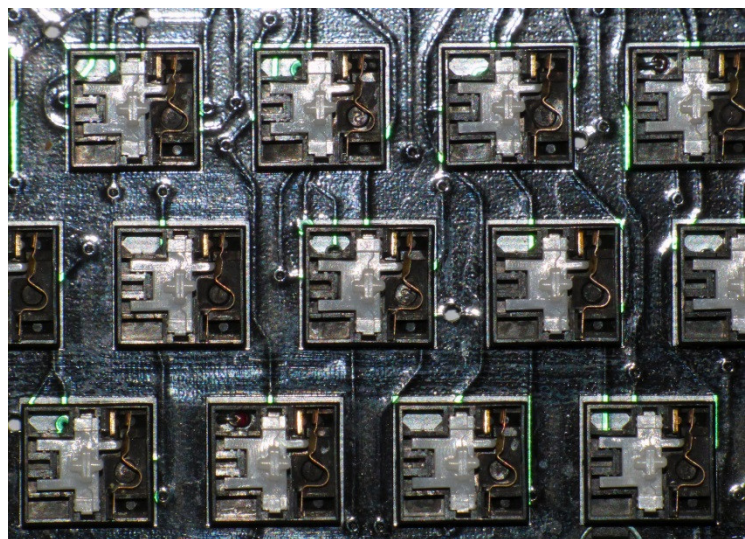
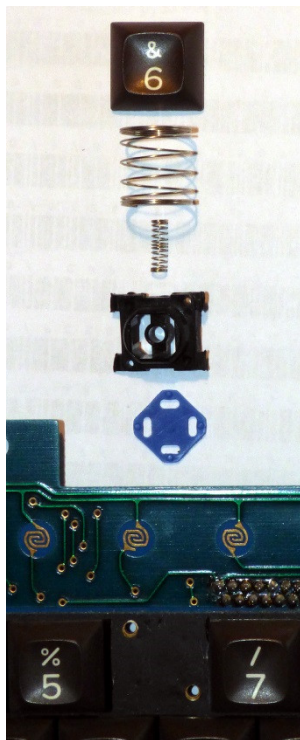


Figure 17: Key '6' assembly used on the membrane keyboard and close-up view of the mechanical counterparts.

12. Loading BASIC Programs via RS-232C

The command

LOAD "COMO:"

can be used to load BASIC programs in text format from a second computer. If you have a Windows system, you can use the RealTerm or Teraterm software to send such files. Without handshaking an inter-character delay of about 10 ms is required to obtain a correct transmission at the default baud rate of 4800.

The sender should terminate the transfer by sending a last character of **CTRL-Z (0x1A)**. Then the **LOAD** command terminates and returns to the command prompt. Otherwise one has to press the BREAK key on the HX-20 to terminate the transfer.

13. Controlling External Devices

The serial interfaces can be used to control any device with a serial interface. If only a simple on/off switching function is required, one can also use the “Remote” output of the HX-20. This connection is intended to control the motor of an external cassette recorder/player. As the schematic shows, it is completely decoupled from the HX-20 electronics by a relay and thus safe to use for external circuits.

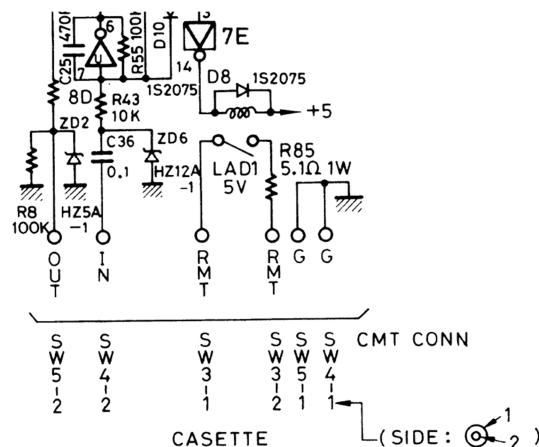


Figure 18: The HX-20 contains a relay to control an external cassette recorder via the REMOTE connector. It can be controlled by the MOTOR command.

The exact specification of this relay is unknown but the schematic shows a voltage of 5 V and a $5.1\ \Omega / 1\ \text{W}$ current limiter resistor. Thus, the current drawn by the external device should never exceed 200 mA – I recommend keeping it below 50 mA at 5 V.

A 2.5 mm mono plug with a small diameter handle is needed for the connection. The small diameter is required for inserting the plug far enough into the HX-20. As I could only find 2.5 mm plugs with a too large diameter of the handle, I soldered the wires and then filled its body with epoxy resin. Finally I used a lathe to turn the diameter of the plastic handle partially down to the required diameter. Alternatively one could also use some silicone rubber or epoxy putty to create a suitable handle.

14. Some Useful Subroutines

14.1. User Defined Characters

The following program fragment can be used to define characters which are assigned to the GRAPH+0 and following keys. It has to be executed only once after a cold start.

```
10 REM Define NCHARS Characters
20 NCHARS=1
30 ADDR=&H0A40
40 MEMSET ADDR+6*NCHARS
50 REM Again, as MEMSET cleared all variables
60 ADDR=&H0A40
70 NCHARS=1
80 LO=ADDR AND &H00FF
90 HI=(ADDR/256) AND &H00FF
100 POKE &H011E,HI
110 POKE &H011F,LO
120 REM NCHARS Character Bitmap(s) of 6 bytes each
130 DATA 92,98,2,98,92,0
140 RESTORE 130
150 FOR N=1 TO 6*NCHARS
160 READ B
170 POKE ADDR,B
180 ADDR=ADDR+1
190 NEXT N
200 STOP
```

14.2. Get the Time in Seconds

By converting the return value of the `TIME$` function we can determine the seconds into the day:

```
210 REM Current Time in Seconds
220 T$=TIME$
230 T#=3600.*CDBL(VAL(MID$(T$,1,2)))
240 T#=T#+60.*CDBL(VAL(MID$(T$,4,2)))
250 T#=T#+CDBL(VAL(MID$(T$,7,2)))
260 RETURN
```

The current time is also maintained in the even memory locations between `0x0040` and `0x0044`. It can be read, converted and displayed by the following code fragment:

```
1000 REM --- TIME ---
1010 T%=0
1020 POKE &H007E,PEEK(&H007E) OR 128
1030 S%=PEEK(&H0040)
1040 M%=PEEK(&H0042)
1050 H%=PEEK(&H0044)
1060 S%=INT((S% AND &F0)/16)*10+(S% AND &H0F)
1070 M%=INT((M% AND &F0)/16)*10+(M% AND &H0F)
1080 H%=INT((H% AND &F0)/16)*10+(H% AND &H0F)
1090 IF S%=T% THEN 1030
1100 IF S%>59 THEN 1030
1110 PRINT USING "##:##:##";H%,M%,S%
1120 PRINT CHR$(&H1E);
1130 T%=S%
1040 T# = 3600.*H% + 60.*M% + CDBL(S%)
1150 GOTO 1030
```

Notes:

- Line 1020 enables access to the low memory region.

- Line 1090 waits to update the display every second.
- Line 1100 catches a problem: the seconds value may be larger than 59, probably when the PEEK in line 1030 occurs just when the clock is updated.
- Line 1120 moves the cursor back to overwrite the time output line.

14.3. Functions to obtain Low and High Byte of an Integer

```
230 DEF FNLO$(X%)=CHR$(X% AND &HFF)
240 DEF FNHI$(X%)=CHR$((X% AND &HFF00)/256)
```

14.4. Decoding a String with a Hexadecimal Number

This is a rather trivial application, but it is easily overlooked that the `VAL` function can do more than parse decimal numbers.

```
230 HX$="0A40"
240 H%=VAL("&H"+HX$)
```

15. Some Benchmark Results

The following table lists some execution times for the infamous BYTE Benchmark “Eratosthenes Primes” [3]. The times given for these roughly comparable systems are all for 10 iterations.

Computer	Year	CPU Type and Speed	Programming Language	Time
HX-20	1982	6301 @ 0.614 MHz	BASIC	4050 s
HX-20	1982	6301 @ 0.614 MHz	Assembler	17 s
HX-20	1982	6301 @ 0.614 MHz	Forth	229 s
TI-99/4	1981	TMS 9900 @ 3.0 MHz	TI-BASIC	3960 s
PET	1977	6502 @ 1.0 MHz	BASIC	3180 s
Apple II	1977	6502 @ 1.02 MHz	Applesoft BASIC	2806 s
HP-85	1980	Capricorn @ 625 kHz	BASIC	3084 s
HP-85	1980	Capricorn @ 625 kHz	Assembler	21 s
TRS-80/II	1977	Z-80 @ 1.77 MHz	MBASIC	2250 s
IBM PC	1981	8088 @ 4.77 MHz	BASICA	1990s

Table 1: Execution times for the BYTE benchmark.

We can clearly see that the HX-20 in BASIC mode is not exactly the fastest computer. In order to restore the honor of this machine I wrote an assembler version of the benchmark. As I had no experience with the 6800 family and the Hitachi 6301, the code is surely not optimized but the results should give a good estimate of what is possible.

```
; -----
; The infamous BYTE Benchmark Eratosthenes Sieve.
; For the Epson HX/20 with Hitachi HS 6301 CPU.
; -----
; This assembly language program performs 10 loops
; of the Sieve benchmark.
; The number of primes is saved in variable "C" at
; address 0x0ADA. The correct result is 1899 (0x076B).
;
; Enter the hex bytes starting at address 0xA40
; using the Monitor.
; Start with
; S0A40
```

		; When the code up to address 0AD3 has been entered, ; it can be executed from 0A40 until the PC reaches ; 0ABE (Label STOP): ; GOA40,0ABE ; ; Assembled from the ASM source with the a09 assembler: ; a09 -oH01 sieve.asm -Lsieve.lst ; ; References: ; BYTE Magazine, January 1983 ; : Created 12/2018 Martin Hepperle ; -----
		OPT H01 ; Hitachi 6301
		ORG \$0A40
Addr Bytes		
0A40 860A		LDA #0A ; 10 times
0A42 B70AD4		STAA REP ; repeat count
		; set FLAG(0:8190)=1
0A45 CC0001	AGAIN	LDD #0001 ; step size=1
0A48 FD0AD9		STD P
0A4B 8601		LDA #01 ; set flag
0A4D B70ADD		STAA F
		; starting address
0A50 CC0AE0		LDD #FLAG ; load address of FLAG, use as...
0A53 FD0ADE		STD FPTR ; ...starting address for FILL
0A56 BD0ABF		JSR FILL ; set *FPTR, *(FPTR+1), ... to F=1
		; preparation of loop
0A59 CC0000		LDD #0000 ; C=0
0A5C FD0AD5		STD C ;
0A5F CCFFFF		LDD #FFFF ; I=-1 for starting loop at 0
0A62 FD0AD7		STD I
0A65 8600		LDA #00 ; clear flag
0A67 B70ADD		STAA F
		; I-loop from 0 to 8190
0A6A FC0AD7	NEXT	LDD I
0A6D C30001		ADDD #00001
0A70 FD0AD7		STD I ; I=I+1
		; compare I against 8191
0A73 18		XGDX ; D->X
0A74 8C1FFF		CPX #1FFF
0A77 273C		BEQ FINI ; end of loop
		; FLAG[I] == 0?
0A79 CC0AE0		LDD #FLAG ; load address of FLAG
0A7C F30AD7		ADDD I ; address of FLAG[I]
0A7F 18		XGDX ; D->X
0A80 A600		LDA \$00,X ; get value from FLAG[I]
0A82 27E6		BEQ NEXT ; if already ZERO: continue I loop
		; I
0A84 FC0AD7		LDD I ; I
0A87 F30AD7		ADDD I ; I+I
0A8A C30003		ADDD #3 ; I+I+3
0A8D FD0AD9		STD P ; P=I+I+3
		; K=P+I
0A90 F30AD7		ADDD I ; K=P+I
0A93 FD0ADB		STD K
		; get C
0A96 FC0AD5		LDD C ; get C
0A99 C30001		ADDD #00001 ; C=C+1

0A9C	FD0AD5		STD	C		; update count of primes
						; K > 8190?
0A9F	FE0ADB		LDX	K		
0AA2	8C1FFE		CPX	#\$1FFE		; 8190
0AA5	2EC3		BGT	NEXT		; continue with loop
						; for J=K to 8190 step P
						; starting address
0AA7	CC0AE0		LDD	#FLAG		; load address of FLAG[K]...
0AAA	F30ADB		ADDD	K		; ...and use as...
0AAD	FD0ADE		STD	FPTR		; ...starting address for FILL
0AB0	BD0ABF		JSR	FILL		; set *(FPTR+K), *(FPTR+K+P), ...
0AB3	20B5		BRA	NEXT		
						; all done, repeat?
0AB5	B60AD4	FINI	LDAA	REP		; get repeat count
0AB8	4A		DECA			; decrement
0AB9	B70AD4		STAA	REP		; store repeat count
0ABC	2687		BNE	AGAIN		; not yet finished
0ABE	39	STOP	RTS			; finally
						; fill FLAG array from *BPTR with F step P
0ABF	FE0ADE	FILL	LDX	FPTR		; address in BPTR = FLAG[J]
0AC2	8C2ADE	LOOP	CPX	#FLGE		; address of last byte in FLAG
0AC5	2E0C		BGT	DONE		; beyond end of FLAG[]: leave loop
0AC7	B60ADD		LDAA	F		; flag value to set (byte)
0ACA	A700		STAA	\$00,X		; insert value into FLAG[J]
0ACC	18		XGDX			; X<->D
0ACD	F30AD9		ADDD	P		; now D has X+P
0AD0	18		XGDX			; bring X+P back to X
0AD1	20EF		BRA	LOOP		; again
0AD3	39	DONE	RTS			; done

0AD4	00	REP	FCB	\$00		
0AD5	0000	C	FDB	\$0000		; prime count, 1899d = 076Bh
0AD7	0000	I	FDB	\$0000		; loop count
0AD9	0000	P	FDB	\$0000		; step size
0ADB	0000	K	FDB	\$0000		; starting index
0ADD	00	F	FCB	\$00		; value to set
0ADE	0000	FPTR	FDB	\$0000		; pointer to array element

		FLAG				; flag array
0AE0	0000000000000000		FILL	\$00,8190		; fill with zero
0AE7	0000000000000000					
	...					
2ADE	00	FLGE	FCB	\$00		; last byte in FLAG array
			END			

16. Writing Machine Language Routines

When I ran the BYTE benchmark "Eratosthenes Sieve" in BASIC, I was disappointed by the low performance. Experience from the HP-85 hinted that writing the code in machine language (using an assembler) could accelerate the program by a huge factor. Therefore, I started looking for ways to write and use assembler programs for the HX-20.

The BASIC Reference Manual contains a brief explanation how to call machine language subroutines with the **EXEC** and **USR** functions. It also explains the structure of BASIC variables so that these can be accessed by machine language programs.

This BASIC interface is rather limited, though: the **EXEC** function does not take any parameters but the **USR** function can take one parameter. Officially, the **USR** function always returns the same type as its parameter, i.e. if the parameter is an integer, the function return type must also be integer (but there is a way to change this by placing the result in the FPACC memory location and by adapting the type information in 0x0085-0x0086). If more than one parameter has to be transferred, these parameters could be copied to predefined global memory locations so that they can be accessed from BASIC as well as from the machine language program. Another option to handle multiple or mixed parameter types is to wrap the parameters into the bytes of a string and write the **USR** function to split this string parameter into its components. Finally for multiple numeric parameters an array could be used.

In Figure 19 I show the register set of the 6301 in comparison to the well-known 6800 and 6809. It can be seen that assembler code for the 6800 should be fairly easy to translate for the 6301. The 6809 has two additional 16-bit registers making a translation less straightforward.

For more information about programming the Hitachi 6301 one should consult the data sheet of the 6301 and books about the 6800 processor family. I could not find any specific book about the 6301, though.

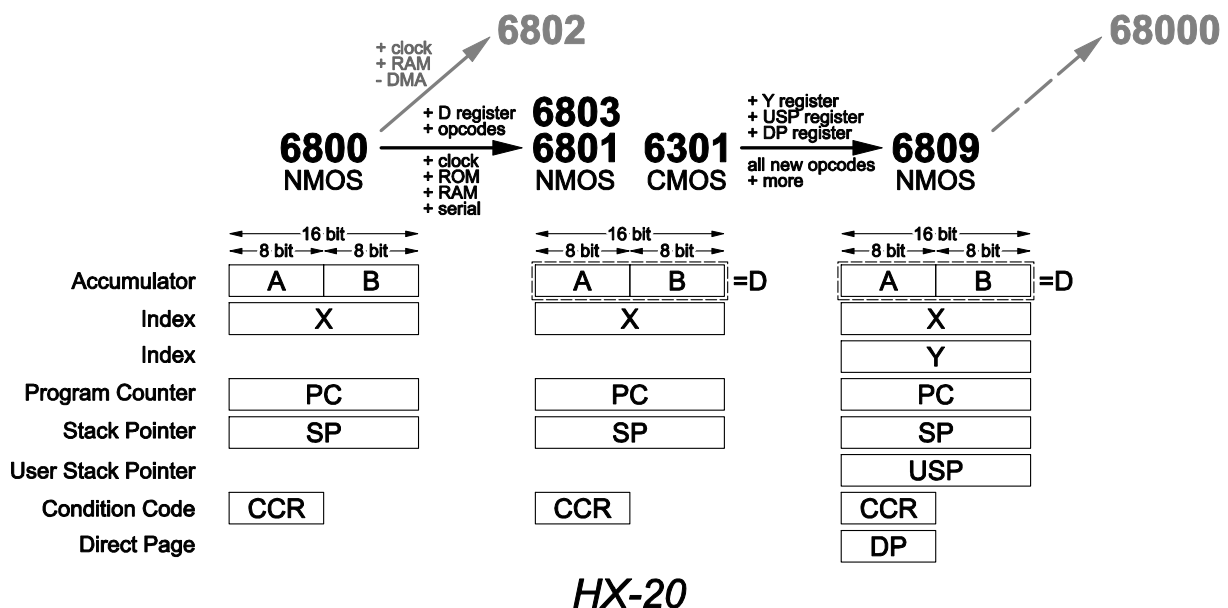


Figure 19: Registers of the 6301 and the related 6800 processor family.

An introduction into the 6301 CPU and its assembler language mnemonics is given in the book by Balkan [4]. It even contains a listing of an assembler written in BASIC and running on the HX-20 or other machines with Microsoft BASIC.

Unfortunately, the listing seems to have been typeset manually so that it contains about a dozen typos as well as one major bug. I used this assembler for my first exploratory steps (after fixing the typos and the bug and running it on a CP/M emulator with MBASIC). However, due to memory limitations of the 16 KB HX-20, this assembler is rather minimalistic.

Therefore, I searched again and found the A09 assembler which had also been extended to cover the 6301 opcodes. This assembler comes in plain "C" and I compiled and executed it on a Windows

system. It can produce listings as well as binary and hexadecimal output. After fixing one bug in its 6301 opcode table it worked fine (by now, the fix should be integrated into the official release).

In order to load the assembled code into the HX-20 I wrote a small Python script which reads the listing file produced by A09 and transforms the code into a BASIC loader program, complete with **MEMSET**, **DATA** and the required **POKE** commands.

The transfer of this BASIC program to the HX-20 is accomplished by the RealTerm program at 4800 baud with an inter-character delay of 5...10 ms.

Thus, my process is

- Connect both machines with the proper RS-232C cable.
- On the PC:
 - assemble the code with **A09**,
 - convert the output into a BASIC loader program using **LST2BAS.py**,
 - set the communication parameters to 4800 baud, 8 bits, no parity, no handshaking and 1 stop bit,
- On the HX-20
 - execute **LOAD "COM0:(68N1E)"** to prepare for loading the program into the HX-20.
- On the PC:
 - use RealTerm to send the BASIC program to the HX-20,
 - wait until the program has been transferred.
- on the HX-20
 - inspect and execute the BASIC program,
 - this last step will actually write the machine code into memory.

After the machine code has been poked into memory, it stays there as long as no **MEMSET** command reduces the amount of reserved memory or another machine code program overwrites this memory. This means that the BASIC loader program has to be run only once. On the other hand, it does not hurt to run it again, if you want to be sure that the memory has not been altered. After loading, the machine code can also be saved to and read from the microcassette using the **SAVEM** respectively the **LOADM** commands. Unfortunately, it seems to be impossible to save and restore binary programs via the RS-232C interface.

If the machine code sequence in the **DATA** statements would become very large, one could modify the loader program to read the **DATA** from the RS-232C port or from tape. So far I wrote only small programs so that this was not necessary and I found it more convenient to keep the machine code together with the loader in a single program.

The Python script:

```
'''
This is a simple tool for converting the listing produced by the
A09 assembler into Epson HX-20 BASIC statements.

The resulting BASIC program loads the machine code into memory.
The code can then be executed by an EXEC statement or by
calling a USR function.

In the DATA statement starting addresses for a range of opcodes
or data are identifiable by a length of four characters.
All opcodes or data bytes are two characters long.
'''

import sys

# -----
```

```

def go ( s ):
    '''
    For Epson HX-20.
    Convert 6301 assembler listing file "s" into BASIC.
    '''

    # grab all (listing files are small)
    fIn = open(s);
    ss = fIn.readlines();
    fIn.close();

    nLines = len(ss);

    # values have to be adapted
    # RETURN is placed
    nStop = 1070

    n = 10
    print (str(n)+' REM --- Epson HX-20      ---')
    n = n+10
    print (str(n)+' REM --- M. Hepperle 2018 ---')
    n = n+10
    print (str(n)+' REM --- adjust BASIC starting address')

    # skip MEMSET line
    n = n+10
    nMemSet = n

    n = n+10
    print (str(n)+' REM --- load the code bytes')
    n = n+10
    print (str(n)+' GOSUB 1010')
    n = n+10
    print (str(n)+' REM --- define the function')
    n = n+10
    nDefFn = n
    n = n+10
    print (str(n)+' REM --- application example')

    # -----
    n = n+10
    print (str(n)+' OPEN "0",#1,"COM0:(68N1D) "')
    n = n+10
    #print (str(n)+' PRINT USR1(CHR$(0)+CHR$(32)+CHR$(0)+CHR$(64)+"Hello World"')')
    print (str(n)+' PRINT USR1("Hello World"')')
    n = n+10
    print (str(n)+' CLOSE #1')
    # -----

    n = n+10
    print (str(n)+' REM --- if no parameters, then use:')
    n = n+10
    print (str(n)+' REM   EXEC &H0A40')
    n = n+10
    print (str(n)+' END')

    n = 1000
    print (str(n)+' REM --- Hex Code Loader  ---')

    n = n+10
    print (str(n)+' N%=0')

    n = n+10
    nLoop = n
    print (str(n)+' READ C$')

    n = n+10
    nReturn = n+50
    print (str(n)+' IF C$="DONE" THEN '+str(nReturn))

    n = n+10
    print (str(n)+' N%=N%+1 : IF N%=8 THEN PRINT ". "; : N%=0')

    # new address, DATA MUST start with an address!
    n = n+10

```

```

print (str(n)+' C%=VAL("&H"+C$)')

# new address, DATA MUST start with an address!
n = n+10
print (str(n)+' IF LEN(C$)=4 THEN A%=C% : GOTO '+str(nLoop))

# new opcode
n = n+10
print (str(n)+' POKE A%,C% : A%=A%+1 : GOTO '+str(nLoop))

# nStop
n = n+10
if n != nReturn:
    print ('*** ERROR: increase nReturn by '+str(n-nReturn))
print (str(n)+' RETURN')

line=0;
address = 0

startAddress = 65536
endAddress = 0

sLine = ''

while line < nLines:
    l = ss[line].replace("\n","")

    # this is the End
    if l.startswith('SYMBOL TABLE'):
        break

    # continuation line has no blank in the first column
    if l[0:1] != ' ':
        #{
            # skip
            line = line+1
            continue
        #}

    addr = l[1:5].strip()

    if len(addr) == 4:
        #{
            try:
                #{
                    addrDec = int(addr,16)

                    if addrDec < startAddress:
                        #{
                            startAddress = addrDec
                        #}

                    if addrDec > endAddress:
                        #{
                            endAddress = addrDec
                        #}

                    if addrDec != address:
                        #{
                            # a step in addresses -
                            # output new start address
                            address = addrDec
                            sLine = sLine + addr + ','
                        #}

                    opcodes = l[6:20].strip()
                    i = 0

                    while i < len(opcodes):
                        #{
                            sLine = sLine + opcodes[i:i+2] + ','
                            i = i+2

                            # update high water mark

```

```

        if address > endAddress:
            #{
                endAddress = address
            #}

        # next free address or start of BASIC for MEMSET
        address = address+1

        if len(sLine)>57:
            #{
                n = n+1
                print (str(n) + ' DATA ' + sLine[0:len(sLine)-1])
                sLine = ''
            #}
        #}
    except:
        #{
            addrDec = 0
        #}
    #}

    line = line+1
    if line > 50000:
        #{
            break
        #}

# flush DATA line
sLine = sLine + 'DONE'
n = n+1
print (str(n) + ' DATA ' + sLine[0:len(sLine)])

# insert MEMSET line above
print (str(nMemSet) + ' MEMSET &H' + hex(endAddress+1).upper()[2:])
print (str(nDefFn)+' DEFUSR1=&H'+hex(startAddress).upper()[2:])

# terminate transfer with ^Z
print ('\032')

if l==1:
    print ('Comments:')
    print ('=====')
    print ('The binary code is loaded into RAM between &H' +
        hex(startAddress).upper()[2:]+ ' and &H' + hex(endAddress).upper()[2:]+'.')
    print ('Thus we need to use MEMSET to shift the start of the BASIC')
    print ('program and data area up:')
    print (' MEMSET &H' + hex(endAddress+1).upper()[2:])
    print ('')
    print ('If the code requires no parameters, you can execute it with')
    print (' EXEC &H' + hex(startAddress).upper()[2:])
    print ('')
    print ('If it takes one parameter, define it as a USR function:')
    print (' DEFUSR1=&H' + hex(startAddress).upper()[2:])
    print ('(Note that USR functions can have only one parameter.)')
    print (' Multiple parameters can often be packed into a string or array')
    print ('Call the function with its parameter and grab the return value:')
    print (' I%=USR1(...parameter...)')
    print ('')
    print ('The generic Hex Code loader at the end of the program reads DATA')
    print ('statements containing either four or two digit hexadecimal numbers.')
    print ('If the number has 4 digits it is interpreted as the "current address".')
    print ('Any following data bytes will be loaded starting at this address.')
    print ('If the number has two digits, it is a data byte which will be loaded')
    print ('to the "current address" and the address is incremented by one.')
    print ('This scheme allows loading data to arbitrary addresses, if desired.')
    print ('Reading the data stream is terminated by the data item DONE.')
    print ('')
    print ('If nothing changes, the data has to be read only once, so that')
    print ('you could add a test for e.g. the first and last bytes and skip loading.')
    print ('The assembler code should usually end with an RTS instruction (&H39).')

# -----

```

```

if __name__ == "__main__":
    if len(sys.argv)>1:
        basePath = "D:\\HP\\Epson HX-20\\ASM\\"
        basePath = './'
        fileName = sys.argv[1]
        go(basePath + fileName)
    else:
        print ('Usage: LST2BAS listing.lst')

```

16.1. Extending the Operating System

The BASIC Operating System of the HX-20 is astonishingly flexible and can be extended in various ways. One option is to add new devices. The core system already supports the devices

KYBD:	keyboard,
SCRN:	screen,
LPT0:	internal printer,
CAS0:	internal cassette (optional),
CAS1:	external cassette,
PACO:	ROM cartridge (optional),
COM0:	RS232C,
BRCD:	barcode reader (optional),
"A:", B:, C:, D:	flexible disk units (optional).

The system allows up to 16 devices. Most devices of these can be accessed sequentially with standard BASIC statements like **OPEN**, **CLOSE**, **PRINT**, **INPUT**, **EOF**, **LOF** and so on.

For each device a device driver is installed in a device table, which is a list of addresses of "Device Control Blocks" (DCB). Each DCB has a prescribed structure and contains the name of the device as well as addresses to a set of required functions. It is possible to add devices by installing a device driver somewhere in RAM and adding the address of its DCB to the table. Before removing the driver, you should uninstall it, by zeroing the address entry in the device table.

The DCB has the following structure:

```

; Device Control Block
DCB:  FCB      "BUF0"          ; 4 character name
      FCB      $30             ; I/O modes: $01: read-only, $20: write-only,
$30: r/w
      FDB      OPENDEV         ; OPEN routine
      FDB      CLOSEDEV        ; CLOSE routine
      FDB      READDEV         ; READ routine
      FDB      WRITEDEV        ; WRITE routine
      FDB      EOFDEV          ; EOF routine
      FDB      LOFDEV          ; LOF routine
DEVBUF: FCB      $00,$00,$00,$00 ; for device purposes
COLPOS: FCB      $00             ; current column position (returned by BASIC
POS function)
MAXCOL: FCB      $00             ; max. column: $00: infinite
PRTTAB: FCB      $14             ; print zone width: step of ",", separated
PRINT output
LSTTAB: FCB      $00             ; last print zone on line
WIDTH:  FCB      $80             ; WIDTH support: $00: yes, $80: no

```

Following the name of the device it contains some flags and the addresses of 6 worker routines. These routines are called when the device is opened or closed, when bytes are read and/or written to the device and when the state of the device is inquired. I/O is performed on a sequential byte-by-byte

basis, so this approach is not the fastest way to read or write large amounts of data. But all devices can be used easily with the standard I/O functions of BASIC.

The following example implements a simple device which provides a small ring buffer of 64 bytes. You can write data to the buffer and later read it back. It is rather useless, but serves as an example of a new device. It is installed with a small BASIC loader program and thus becomes available for all BASIC LOGINS. I decided to place it just below BASIC into RAM starting at address \$0A40. This was the most convenient option for me.

To generate such a driver you

- write the assembler code, starting with the DCB and implementing the required functions,
- assemble the code with a suitable assembler (I use a09),
- translate the binary bytes into BASIC DATA statements and append it to the BASIC loader program (I use a python script to read the listing produced by a09),
- run the BASIC loader program, which
 - moves the starting point of BASIC up to leave room for the new device code,
 - loads the DCB and the code into the free space,
 - installs the address of the DCB in the device table.

After the driver has been installed, it can be used until it is removed from the device table by replacing the address of its DCB in the device table with null bytes. This removal is also implemented in the example BASIC program.

When the driver has been loaded once, it would also be possible to SAVEM it to a disk or tape file. Later, a BASIC program would only need the correct MEMSET and a LOADM command would replace the slow HEX loader. The MEMSET command is important to keep the allocated space from being overwritten by BASIC (there are also other but more complex ways to reserve space for such a driver).

```
; a09 device.asm -ldevice.lst
; python LST2BAS.py device.lst > device.bas

; HX-20: Hitachi 6301
OPT H01
OPT NCL

; insert below BASIC into the address range $0A40...$0AF7
ORG      $0A40

;
; A device driver skeleton for the Epson HX-20
;
; Implements a simple device "BUF0:" which stores
; bytes written to it in a ring buffer.
; Reading from the device returns the bytes written until
; the buffer is empty.
; The LOF() function returns the amount of data currently in the buffer.
; The EOF() function returns 0 if there is something in the buffer,
; -1 otherwise.
;
; Example
;
; 10 REM --- Epson HX-20      ---
; 20 REM --- M. Hepperle 2024 ---
; 30 REM --- adjust BASIC starting address
; 40 MEMSET &HAF8;
```



```

; 50 REM --- load the device "BUF0"
; 60 GOSUB 350
; 70 REM --- application example
; 80 OPEN "O",#1,"BUF0:"
; 90 PRINT #1,"123";
; 100 PRINT "POS(1)=";POS(1)
; 110 PRINT #1,"ABC"
; 120 PRINT #1,"abc";
; 130 PRINT "POS(1)=";POS(1)
; 140 CLOSE #1
; 150 ON ERROR GOTO 240
; 160 OPEN "I",#1,"BUF0:"
; 170 PRINT "Buffer size=";LOF(1)
; 180 FOR I%=1 TO 100
; 190 C$=INPUT$(1,#1)
; 200 IF ASC(C$)=13 THEN C$="CR"
; 210 IF ASC(C$)=10 THEN C$="LF"
; 220 PRINT "/" ; C$;
; 230 NEXT I%
; 240 CLOSE #1
; 250 PRINT "/"
; 260 PRINT "UNLINK BUF0:"
; 270 INPUT "Y/N";YN$
; 280 IF YN$<>"Y" THEN GOTO 330
; 290 IF A%=0 THEN GOTO 330
; 300 POKE A%,0
; 310 POKE A%+1,0
; 320 PRINT "BUF0: at";A%;"removed"
; 330 END
; 340 REM --- Hex Code Loader ---
; 350 N%=0
; 360 READ C$
; 380 IF C$="DONE" THEN 430
; 390 N%=N%+1 : IF N%=8 THEN PRINT "."; : N%=0
; 400 C%=VAL("&H"+C$)
; 410 IF LEN(C$)=4 THEN A%=C% : GOTO 360
; 420 POKE A%,C% : A%=A%+1 : GOTO 360
; 430 PRINT
; 440 REM install device control block in DCB table
; 450 DCBTAB%=&H0657
; 460 FOR A%=DCBTAB% TO DCBTAB%+30 STEP 2
; 470 C%=PEEK(A%)*256+PEEK(A%+1)
; 480 IF C%=&HA40 THEN GOTO 540
; 490 IF C%=&H000 THEN GOTO 520
; 500 NEXT A%
; 510 IF A%>DCBTAB%+28 THEN GOTO 560 : REM ERROR
; 520 POKE A%,&H0A : REM HIGH
; 530 POKE A%+1,&H40 : REM LOW
; 540 PRINT "BUF0: at";A%;"added" : REM remember A% for later removal
; 550 RETURN
; 560 PRINT "Cannot install BUF0:"
; 570 STOP
; 580 DATA 0A40,42,55,46,30,30,0A,5E,0A,5F,0A,60,0A,7C,0A,A0,0A,AC,00
; 590 DATA 00,00,00,00,00,14,00,80,0A,B8,0A,B8,39,39,FE,0A,5A,BC,0A,5C
; 600 DATA 27,0F,A6,00,08,8C,0A,F8,26,03,CE,0A,B8,FF,0A,5A,39,86,FF,97
; 610 DATA F5,39,5F,D7,F5,FE,0A,5C,A7,00,08,8C,0A,F8,26,03,CE,0A,B8,FF
; 620 DATA 0A,5C,81,0D,27,08,81,0A,27,04,7C,0A,55,39,7F,0A,55,39,5F,FE
; 630 DATA 0A,5A,BC,0A,5C,27,01,39,5A,39,FC,0A,5C,B3,0A,5A,2A,03,C3,00
; 640 DATA 40,39,DONE
; 650 REM --- END

; EOFLG EQU $00F8 ; EOF flag (error in Epson manual)
EOFLG EQU $00F5 ; EOF flag (J. Wald and system ROM A000-BFFF)

; -----
; Device Control Block
DCB: FCB "BUF0" ; 4 character name

```

```

FCB      $30          ; I/O mode: $01: r, $20: w, $30: r/w
FDB      OPENDEV      ; OPEN routine
FDB      CLOSEDEV     ; CLOSE routine
FDB      READDEV      ; READ routine
FDB      WRITEDEV     ; WRITE routine
FDB      EOFDEV       ; EOF routine
FDB      LOFDEV       ; LOF routine
DEVBUF: FCB      $00,$00,$00,$00 ; for device purposes
COLPOS: FCB      $00          ; current column position see BASIC POS(#)
MAXCOL: FCB      $00          ; max. column: $00: infinite
PRTTAB: FCB      $14         ; print zone width: step of ",", PRINT output
LSTTAB: FCB      $00          ; last print zone on line
WIDTH: FCB      $80          ; WIDTH support: $00: yes, $80: no
; -----
; max. 64 bytes
READPT FDB      BUFFER      ; initialize read and write addresses
WRITPT FDB      BUFFER      ; initially buffer is empty
; -----
; called by OPEN
; OPEN "I",#1,"BUF:"
; OPEN "O",#1,"BUF:"
OPENDEV
; no action
RTS
; -----
; called by CLOSE
; CLOSE #1
CLOSEDEV
; no action
RTS
; -----
; called e.g. by INPUT$
; C$=INPUT$(n,#1)
; read one byte from device
; return byte in (A) or set EOF flag to $FF on EOF
READDEV
LDX      READPT          ; get read address
CPX      WRITPT          ; compare with write position
BEQ      READ_EOF        ; buffer is empty

LDAA     ,X              ; get byte from buffer
INX      ; increment pointer
CPX      #BUFEND          ; get address
BNE      READ_1          ; o.k.
; wrap
LDX      #BUFFER          ; back to start
READ_1
STX      READPT          ; for next read

RTS

READ_EOF
LDAA     #$FF            ; EOF: $FF
STAA     EOFLG           ;
RTS
; -----
; called e.g. by PRINT#
; PRINT# 1,"ABC"
; write one byte to device
; (A) byte to write
WRITEDEV
CLRB      ; not at EOF: $00

```

```

        STAB    EOFLG    ;

        LDX     WRITPT           ; get current write address
        STAA    ,X              ; store byte
        INX
        CPX     #BUFEND         ; get address
        BNE     WRITE_1

        ; buffer overflow: wrap
        LDX     #BUFFER         ; get start address

WRITE_1
        STX     WRITPT           ; for next write
        ; increment or reset POS
        CMPA    #$0D
        BEQ     ZERPOS
        CMPA    #$0A
        BEQ     ZERPOS
INCPOS   INC     COLPOS    ; increment column index
        RTS
ZERPOS   CLR     COLPOS    ; reset column index
        RTS

; -----
; called e.g. by EOF(1)
EOFDEV
        CLRB                     ; not at EOF
        LDX     READPT           ; get read address
        CPX     WRITPT           ; compare with write position
        BEQ     EOF              ; buffer is empty
        RTS

EOF
        DECB                     ; return $FF EOF flag
        RTS

; -----
; called e.g. by LOF(1)
; return # of bytes in buffer

; wrapping example/test case with 7 byte buffer
; [.ABCDE.].      --- ABCDE not wrapped
; [1234567]8      E=end,   behind buffer      W write pointer
; [BR...W]E       B=begin, buffer              R read pointer
;   W>R: LOF = (W-R) = (7-2) = 5
;
; [E..ABCD].      --- ABCDE wrapped around
; [1234567]8
; [BW.R...]E
;   R>W: LOF = (E-R)+(W-B) = (8-4)+(2-1) = 4 + 1 = 5
;           = (W-R)+(E-B) = (2-4)+(8-1) = -2 + 7 = 5
;           W-R is negative, must add buffer length
LOFDEV
        LDD     WRITPT           ; get read address
        SUBD    READPT
        BPL     LOF_1           ; positive: no wrap
        ADDD    #(BUFEND-BUFFER) ; wrap
LOF_1
        RTS

; -----
; placing the I/O buffer at the end allows omitting these bytes from loading
BUFFER   FILL   $00,64          ; buffer of 64 bytes
BUFEND
MEMSET   $*                    ; same as BUFEND, first free address, for MEMSET

END

```

A similar example is "STAT:", a device which counts the occurrence of each character sent to it. After installing it, one can LIST "STAT:" a program to it and later sequentially read the number of times each character occurred in the listing. For this purpose the device maintains a buffer of 256 16-bit counters, which can be read byte by byte. Thus, opening it for input and reading the first two bytes (high, low) yields the count of CHR\$(0) sent to the device, CHR\$(1) is returned in the next two bytes and so on until the last bytes 511 and 512 define the number of occurrences of CHR\$(255). Each OPEN for output resets the counting array, an OPEN for input resets the reading index to the start of the array.

The example application just sends 3 A's, 2 B's and one C as well as a CHR\$(13) and CHR\$(10) at the end of the PRINT statement. After running it once, you can directly call RUN 150 e.g. after listing a program with LIST "STAT:".

```

10 REM --- Epson HX-20 ---
20 REM --- M. Hepperle 2024 ---
21 REM Installs a device "STAT:".
22 REM Writing to it counts the
23 REM number of occurrences of
24 REM each character code (0...255).
25 REM Reading returns 256*2 bytes
26 REM (high,low) which represent the
27 REM accumulated number of each character.
28 REM One can use LIST "STAT:" to count
29 REM how often each character occurs.
30 REM --- adjust BASIC starting address
40 MEMSET &HCB3
50 REM --- load the code bytes
60 GOSUB 350
70 REM --- install driver
80 GOSUB 450
90 REM --- application example
100 OPEN "O",#1,"STAT:"
110 FOR I%=1 TO 150
120 PRINT#1,CHR$(0)+"AAABBC"
130 NEXT I%
140 CLOSE #1
145 REM --- entry e.g. after LIST "STAT:"
150 PRINT "Finding maximum..."
160 MX=0
170 OPEN "I",#1,"STAT:"
180 FOR I%=0 TO 255
190 H$=INPUT$(1,#1) : L$=INPUT$(1,#1)
200 Y=ASC(H$)*256+ASC(L$)
210 IF Y>MX THEN MX=Y
220 NEXT I%
230 CLOSE #1
240 GCLS
250 OPEN "I",#1,"STAT:"
260 FOR I%=0 TO 127
270 H$=INPUT$(1,#1) : L$=INPUT$(1,#1)
280 Y%=31*(1-(ASC(H$)*256+ASC(L$))/MX)
290 LINE(I%,31)-(I%,Y%),PSET
300 NEXT I%
310 CLOSE #1
320 SOUND 33,2
330 C$=INPUT$(1)
340 END
350 REM --- Hex Code Loader ---
360 N%=0
370 READ C$
380 IF C$="DONE" THEN 430

```

```

390 N%=N%+1 : IF N%=8 THEN PRINT "."; : N%=0
400 C%=VAL("&H"+C$)
410 IF LEN(C$)=4 THEN A%=C% : GOTO 370
420 POKE A%,C% : A%=A%+1 : GOTO 370
430 PRINT
440 RETURN
450 REM --- Device Installer ---
460 DCBTAB%=&H0657
470 FOR A%=DCBTAB% TO DCBTAB%+30 STEP 2
480 C%=PEEK(A%)*256+PEEK(A%+1)
490 IF C%=&H0A40 THEN GOTO 550
500 IF C%=&H0000 THEN GOTO 520
510 NEXT A%
520 IF A%>DCBTAB%+28 THEN GOTO 570
530 POKE A%,&H0A : REM HIGH
540 POKE A%+1,&H40 : REM LOW
550 PRINT "STAT: @";A%;"installed"
560 RETURN
570 PRINT "Cannot install STAT:"
580 STOP
2141 DATA 0A40,53,54,41,54,30,0A,5A,0A,78,0A,79,0A,8F,0A,9F,0A,AB,00
2142 DATA 00,00,00,00,00,14,00,80,B6,06,8A,81,10,27,10,CE,0A,B1,86,FF
2143 DATA 6F,00,6F,01,08,08,4A,81,FF,26,F5,CE,0A,B1,FF,0C,B1,39,39,7F
2144 DATA 00,F5,FE,0C,B1,8C,0C,B1,27,07,A6,00,08,FF,0C,B1,39,7A,00,F5
2145 DATA 39,36,33,4F,05,C3,0A,B1,18,EC,00,F3,0A,AF,ED,00,39,5F,FE,0C
2146 DATA B1,8C,0C,B1,27,01,39,5A,39,CC,00,02,39,00,01,DONE

```

Assembler source code:

```

10 REM --- Epson HX-20 ---
20 REM --- M. Hepperle 2018 ---
21 REM Installs a device "STAT:".
22 REM Writing to it counts the
23 REM number of occurrences of
24 REM each character code (0...255).
25 REM Reading returns 256*2 bytes
26 REM (high,low) which represent the
27 REM accumulated number of each character.
28 REM One can use LIST "STAT:" to count
29 REM how often each character occurs.
30 REM --- adjust BASIC starting address
40 MEMSET &HCB3
50 REM --- load the code bytes
60 GOSUB 350
70 REM --- install driver
80 GOSUB 450
90 REM --- application example
100 OPEN "O",#1,"STAT:"
110 FOR I%=1 TO 300
120 PRINT#1,CHR$(0)+"AAABBC"
130 NEXT I%
140 CLOSE #1
145 REM --- entry e.g. after LIST "STAT:"
150 PRINT "Finding maximum..."
160 MX=0
170 OPEN "I",#1,"STAT:"
180 FOR I%=0 TO 255
190 H$=INPUT$(1,#1) : L$=INPUT$(1,#1)
200 Y=ASC(H$)*256+ASC(L$)
210 IF Y>MX THEN MX=Y
220 NEXT I%
230 CLOSE #1
240 GCLS
250 OPEN "I",#1,"STAT:"
260 FOR I%=0 TO 127
270 H$=INPUT$(1,#1) : L$=INPUT$(1,#1)
280 Y%=31*(1-(ASC(H$)*256+ASC(L$))/MX)

```

```

290 LINE(I%,31)-(I%,Y%),PSET
300 NEXT I%
310 CLOSE #1
320 SOUND 33,2
330 C$=INPUT$(1)
340 END
350 REM --- Hex Code Loader ---
360 N%=0
370 READ C$
380 IF C$="DONE" THEN 430
390 N%=N%+1 : IF N%=8 THEN PRINT ". "; : N%=0
400 C%=VAL("&H"+C$)
410 IF LEN(C$)=4 THEN A%=C% : GOTO 370
420 POKE A%,C% : A%=A%+1 : GOTO 370
430 PRINT
440 RETURN
450 REM --- Device Installer ---
460 DCBTAB%=&H0657
470 FOR A%=DCBTAB% TO DCBTAB%+30 STEP 2
480 C%=PEEK(A%)*256+PEEK(A%+1)
490 IF C%=&H0A40 THEN GOTO 550
500 IF C%=&H0000 THEN GOTO 520
510 NEXT A%
520 IF A%>DCBTAB%+28 THEN GOTO 570
530 POKE A%,&H0A : REM HIGH
540 POKE A%+1,&H40 : REM LOW
550 PRINT "STAT: @";A%;"installed"
560 RETURN
570 PRINT "Cannot install STAT:"
580 STOP
2141 DATA 0A40,53,54,41,54,30,0A,5A,0A,78,0A,79,0A,8F,0A,9F,0A,AB,00
2142 DATA 00,00,00,00,00,14,00,80,B6,06,8A,81,10,27,10,CE,0A,B1,86,FF
2143 DATA 6F,00,6F,01,08,08,4A,81,FF,26,F5,CE,0A,B1,FF,0C,B1,39,39,7F
2144 DATA 00,F5,FE,0C,B1,8C,0C,B1,27,07,A6,00,08,FF,0C,B1,39,7A,00,F5
2145 DATA 39,36,33,4F,05,C3,0A,B1,18,EC,00,F3,0A,AF,ED,00,39,5F,FE,0C
2146 DATA B1,8C,0C,B1,27,01,39,5A,39,CC,00,02,39,00,01,DONE

```

16.2. Some Details about HX-20 BASIC (Microsoft BASIC)

16.2.1. The Floating Point Accumulator

Microsoft BASIC maintains a so called “floating point accumulator” (FPACC). This is a memory area used for intermediate results when working with 16-bit integer as well as single and double floating point numbers. It is also used to transfer a numeric parameter to a `USR` function. Its length is 8 bytes to hold a double precision floating point number. The arrangement of the bytes can be found in the BASIC reference manual. The location of the FPACC is at address `0x00D5` in RAM.

16.2.2. Memory allocation of Arrays

Allocation of a one-dimensional `INTEGER` array:

```

DIM N%(5)
A%=VARPTR(N%(0))

```

The `VARPTR` function returns the address of the first array element `(0)`. In memory this is followed the next element `(1)`.

Allocation of a two-dimensional `INTEGER` array:

```

DIM N%(5,6)

```



```
A%=VARPTR(N%(0,0))
```

The **VARPTR** function returns the address of the first array element (0,0). In memory this is followed the next element (1,0), i.e. the first index is incremented first.

Note: the examples above use the default **OPTION BASE 0** setting. If **OPTION BASE 1** is used, the first element is (1), respectively (1,1).

16.2.3. The BASIC Work Areas

Work Area (1)

Example memory dump:

```
00000080 00 22 00 00 00 04 00 00 00 00 00 08 6C 08 69 1D .".....l.i.
0085-0086:          <--->          TypeInfo for data in FPACC
00000090 7E 1D 80 00 00 00 00 0A 00 00 7D 65 0B 0C 1D 7C ~.....}e...|
009C-009D:          <---> HeadPointer:
                        address of address-
                        field of first line
009E-009F:          <---> StringSpace:
                        address of
                        string space
000000A0 1D 84 1D 84 7D 89 7E 51 7E 51 7E 51 07 DA 07 DA ....}.~Q~Q~Q....
00A0-00A1:<---> NextFree? address of next free entry in string space
00A2-00A3:          <---> NextFree? address of next free entry in string space

000000B0 07 D0 00 00 10 8D 00 00 0B 0B 00 00 01 1D 7E 1D .....~.
00B8-00B9:          <---> DataPointer: address of separator
                        of next line for READ
00BA-00BB:          <---> TailPointer: address of last
                        line (after program was run)
000000C0 82 00 00 00 1D 84 1D 80 00 00 00 00 00 04 BD .....
000000D0 00 00 00 00 00 88 00 00 D8 00 00 00 00 00 01 .....
00D5-00DC:          <----- FPACC -----> Floating Point
                        Accumulator
000000E0 7E 51 00 00 00 00 E6 00 00 00 00 00 00 10 76 ~Q.....v
000000F0 1D 7D 08 5D 00 00 0E 00 5F 00 B6 10 9B 7E B3 D8 .}.].....~..
```

Work Area (2)

Example memory dump:

```
000005B0 00 00 00 88 DF 00 00 00 88 DF 00 00 00 00 00 00 .....
000005C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000005D0 00 00 00 00 00 00 00 00 B4 F3 00 00 00 00 00 00 .....
000005E0 00 00 7E 8C 70 7E 8C 70 7E 8C 70 7E A6 71 7E A6 ..~.p~.p~.p~.q~.
          <---0--> <---1--> <---2--> <---3--> <---- 39 error handlers
000005F0 71 7E A6 71 7E A6 71 7E A6 71 7E A6 71 7E A6 71 q~.q~.q~.q~.q~.q
-> <-----> <-----> <-----> <-----> <----->
00000600 7E A6 71 7E A6 71 7E A6 71 39 A6 71 39 A6 71 39 ~.q~.q~.q9.q9.q9
<-----> <-----> <-----> <-----> <-----> <-
00000610 A6 71 39 A6 71 39 A6 71 39 A6 71 39 A6 71 39 A6 .q9.q9.q9.q9.q9.
-----> <-----> <-----> <-----> <-----> <----
00000620 71 39 A6 71 39 A6 71 39 A6 71 39 A6 71 39 A6 71 q9.q9.q9.q9.q9.q
-> <-----> <-----> <-----> <-----> <----->
00000630 39 A6 71 39 A6 71 39 A6 71 39 A6 71 39 A6 71 39 9.q9.q9.q9.q9.q9
<-----> <-----> <-----> <-----> <-----> <-
00000640 A6 71 39 A6 71 39 A6 71 39 A6 71 7E 88 DF 7E 88 .q9.q9.q9.q~...~.
-----> <-----> <-----> <-----> <-----> <----
00000650 DF 7E 88 DF 7E 88 DF 06 9C 06 B6 06 D0 06 EA 07 ~..~.....
-> <--37--> <--38--> <-0--> <-1--> <-2--> <-3--> <- 16 DCB Addresses
00000660 04 07 1E 07 38 00 00 00 00 00 00 00 00 00 00 00 ....8.....
-> <-5--> <-6--> <-7--> <-8--> <-9--> <-0--> <-1--> <-
```

```

00000670 00 00 00 00 00 00 00 00 22 00 00 00 00 00 00 .....".
-> <-3-> <-4-> <-5->
00000680 00 00 00 00 00 00 00 00 00 00 20 01 FF 03 02 00 .....
00000690 38 4E 31 45 00 00 00 00 8C 70 00 00 4B 59 42 44 8N1E....p..KYBD
069C: <-----> Name of D0
000006A0 10 B3 E4 B3 E4 A9 69 B7 30 B3 E4 B3 E4 B3 E4 B3 .....i.0.....
000006B0 E4 00 00 00 00 80 53 43 52 4E 20 B3 E4 B3 E4 B3 .....SCRN .....
06B6: <-----> Name of D1
000006C0 E4 B7 30 B3 E4 B3 E4 B3 E4 00 00 00 28 0E 1C 80 ..0.....(...
000006D0 43 4F 4D 30 30 B0 6A B0 B3 B0 43 B1 11 B0 13 B0 COM00.j...C.....
06D0: <-----> Name of D2
000006E0 0D B1 28 B1 28 04 00 0E 00 40 43 41 53 30 30 AD ..(.(...@CAS00.
06EA: <-----> Name of D3
000006F0 8D AD DA AE 32 AE 70 AE 7C B3 E4 FE F8 00 00 00 ....2.p.|.....
00000700 00 01 00 C1 43 41 53 31 30 AD 8D AD DA AE 32 AE ....CAS10.....2.
0704: <-----> Name of D4
00000710 70 AE 7C B3 E4 FF 31 00 00 00 00 01 00 C1 50 41 p.|...1.....PA
071E: <-----> Name of D5
00000720 43 30 10 B1 95 B2 00 B2 09 B1 28 B0 1F B0 1B B1 C0.....(.....
-----> Name of D5 cont.
00000730 28 00 00 00 00 00 00 C0 4C 50 54 30 20 B3 E4 B0 (.....LPT0 ...
0738: <-----> Name of D6
00000740 24 B3 E4 B0 2F B3 E4 B3 E4 B3 E4 B3 E4 00 18 0E $.../.....
00000750 0E 00 00 FF 07 D5 88 20 32 30 30 30 00 00 00 00 ..... 2000....
00000760 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00000770 20 20 20 20 20 20 20 20 20 20 20 20 20 20 29 )
00000780 00 44 2C 42 44 2C 46 46 2C 37 30 2C 33 39 2C 46 .D,BD,FF,70,39,F
00000790 46 2C 46 46 2C 30 30 00 00 00 00 00 00 00 00 00 F,FF,00.....
000007A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000007B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000007C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000007D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000007E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000007F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000810 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000820 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000830 09 08 38 08 08 38 F4 6C 06 02 86 00 00 14 00 00 ..8..8.1.....
00000840 DE 41 D9 43 D8 5D 8C 22 08 E7 D7 C2 08 E7 FF B6 .A.C.].".
00000850 EF 7F 02 00 00 B3 58 00 00 00 00 00 41 25 00 00 .....X.....A%..
00000860 00 00 00 00 00 00 00 00 00 00 00 00 01 7E 51 00 .....~Q.
00000870 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000880 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 .....
00000890 00 00 00 00 00 80 00 00 10 B3 00 07 DA 00 0A 1D .....
000008A0 65 1D 7C 07 D5 00 00 00 00 05 07 DA 7D 75 00 10 e.|.....}u..
000008B0 8D 00 00 10 9A 04 04 04 04 04 04 04 04 04 04 .....
08B5- <----- type code table A-Z -----
000008C0 04 04 04 04 04 04 04 04 04 04 04 04 04 04 8C .....
-08CE:-----> <- 10 addresses of
08CF- USR0-9 functions
default 8C70: in BASIC ROM
000008D0 70 8C 70 8C 70 8C 70 8C 70 8C 70 8C 70 8C p.p.p.p.p.p.p.p.
000008E0 70 8C 70 00 00 00 00 92 00 06 00 20 20 20 20 p.p.....
-08E2:----->
000008F0 20 20 20 20 20 20 00 00 00 00 00 00 00 00 00 .....
00000900 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000910 06 D0 00 00 00 00 00 00 00 00 00 20 F9 68 00 00 .....h..
00000920 26 00 00 00 00 00 00 00 00 00 00 00 00 00 00 &.....
00000930 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000940 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000950 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000960 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000970 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000980 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000990 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000009A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000009B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

000009C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000009D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000009E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000009F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000A00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000A10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000A20	00	00	00	00	00	00	00	FF	00	14	04	08	00	00	00	00	00
00000A30	00	00	01	00	00	39	00	00	01	00	00	01	01	00	00	01	019.....

17. Using a Printer

An Epson P-40 printer (or any other printer with serial interface) can easily be connected to the RS-232C port of the HX-20. However, as the buffer of the P-40 is only 2 bytes, data transfer will only work properly if you wire the cable for hardware handshaking. This requires the connection of the printer handshake signal DTR to the HX-20 input signal DSR on pin 6 of the DIN connector.

You can then use commands like

```
LIST "COM0:(68N2B)"
```

to list a program on a printer set to 4800 baud and 8 data bits, no parity and two stop bits

Similarly, the statement

```
OPEN "O",#1,"COM0:(68N2B)"
```

can be used in a program to open a file for output with subsequent **PRINT #1** statements. When done with printing, you should close the serial port with a **CLOSE #1** statement.

18. MH-20 – A Peripheral Emulator

The “MH-20” software runs on a PC and mimics two different peripherals for the HX-20:

- a display controller for text and graphics output and,
- a disk drive units with four disk drives (which equals two TF-20 drives).

While the display function is readily available with the HX-20, the disk drive emulation requires the setting of the switch SW4 to the ON position. This switch is accessible from the bottom of the HX-20. See the “Operating Manual”, page 2-1.

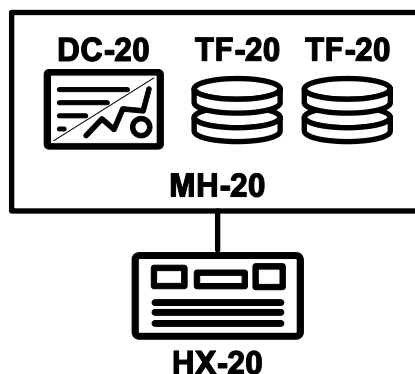


Figure 20: Schematic of the HX-20 with the MH-20 software.

18.1. Required Hardware for HX-20

The MH-20 program listens on the serial RS-232C port of your computer which must be connected to the high speed serial port of the HX-20. The emulator sets the serial port on the PC side to 38400 baud, 8 data bits, 1 stop bit, no parity and no handshaking. The wiring of a cable connecting the HX-20 with a standard IBM-AT-style D-SUB 9 pin male connector is shown in Figure 21. The common USB-RS-232C converter cables usually come with a matching male D-SUB connector and can be used.

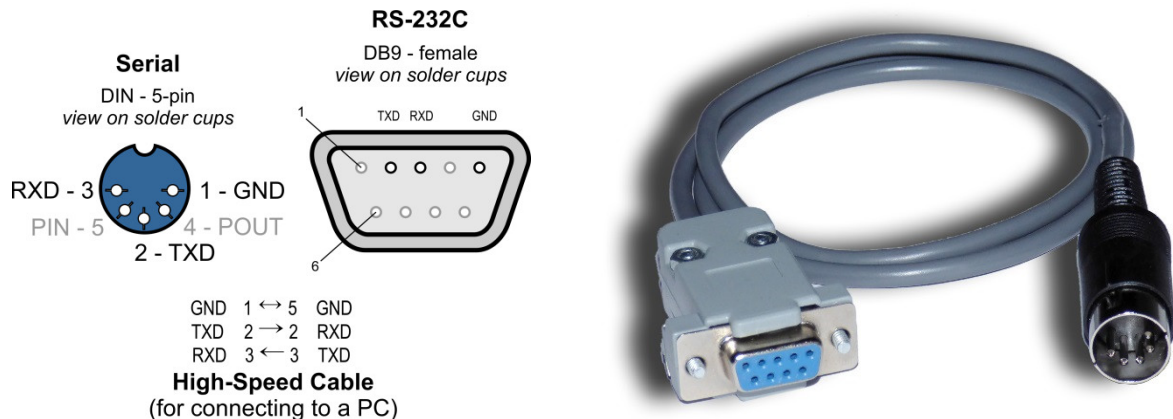


Figure 21: Cable to connect to HX-20 to a PC running the MH-20 screen and disk emulator. Only 3 wires are needed.

18.2. Using the MH-20 Software

MH-20 is written in Java and therefore is executable on many common platforms. You need a Java Runtime Environment (JRE) of Version 1.8 or higher. For the serial communication it relies on the jSSC (Java Simple Serial Connector) serial port communication library. This library includes system dependent hardware drivers for Linux, Mac OS/X, Solaris and Windows 32 as well as Windows 64.

You can start the simulator from a command line and supply these optional command line arguments:

- **-port PORT**
default: PORT=COM1
The device name of your serial port. You must use the proper syntax for your operating system, e.g. for higher port numbers under Windows: “/././COM38”, omit any trailing colon.
- **-width WIDTH**
The width of the window in character columns. Default: WIDTH=80
- **-height HEIGHT**
The height of the window in character rows. Default: HEIGHT=48
- **-diskconfig TYPE**
The arrangement of disk drives. Use TYPE=0 for HX-20 (you can use the emulator also for the PX-8 and for this application other configurations are available)
- **-debug**
Activates output of debug information.

In a Windows command prompt you can enter a command line for the HX-20 like

```
java -jar MH-20-Display-Controller.jar -port ../../COM38 -width 80 -height 24
```

Of course you can and should wrap this long command into a `.cmd` script file.

Under Linux you might have the problem that the serial port is usually not accessible by normal users. You have to be a super-user to work with it. Two options to handle this problem are listed below.

- Create a shell script (text file) e.g. “mh20dc.sh” with the desired command line options. Port access may require administrator rights. Therefore, you can use `sudo` which asks for the superuser password.

```
#!/bin/sh
sudo java -jar MH-20-Display-Controller.jar -port /dev/ttyS0
```

or

- You can also make your script file “mh20dc.sh” set the superuser-bit by itself:

```
sudo chmod +s mh20.sh
```

Then your script would need no `sudo` command, but just the command line

```
#!/bin/sh
java -jar MH-20-Display-Controller.jar -port /dev/ttyS0
```

In both cases you can run the program by executing your script

```
./mh20.sh
```

18.3. Display Controller Emulation

The MH-20 program mimics an external display controller similar to the ones which were available in its day. One such device was the Oval HO-80 from Oval Ltd., a British company, which delivered its video output in form of UHF or PAL signals. Its screen was able to show 32×16 characters or 128×64 pixels in 4 colors or 128×96 pixels in monochrome.

My goal was not a faithful representation of this device (which I even don’t own) and its limitations but mainly to allow for easier reading and editing of programs for the HX-20. Editing programs on the small built-in LCD screen is not really fun – at least for me.

The HX-20 display system supports two operating modes: text mode and graphics mode. Both are partially implemented in the MH-20 software. The text mode offers all cursor movement and editing functions. The special graphics characters are also displayed, but no attempt has been made to implement user defined characters. I even don’t know whether the original display controller was able to handle those.

After the text mode worked sufficiently well for practical application I added some of the graphics functions. These allow clearing the screen (`GCLS`), drawing lines (`LINE`) and setting points (`PSET`) and inquiring the color of pixels (`POINT`).

Like with the original display controller, graphics and text screen are handled as exclusive entities. The MH-20 is either in text or in graphics mode - you cannot mix graphics and text.

However, to allow writing text in graphics mode I implemented an additional command to write a string of characters to the graphics screen. However, this requires the usage of a machine code subroutine to send out the proper data frames.

18.3.1. Applicable BASIC Keywords and Commands

Selecting the Output Device	Purpose
SCREEN 1,0	Send subsequent <i>text output</i> to the display controller.
SCREEN 0,1	Send subsequent <i>graphics output</i> to the display controller.
SCREEN 0,0	Send all subsequent output to the LCD display.

The **SCREEN** command also selects the character set according to the current system settings.

Text Mode	Purpose
CLS	Clear the screen.
PRINT	Print output to the screen.
LIST	List the current program on the screen.
WIDTH width,height	Set the dimensions of the text screen in character cells.
POS	Return the horizontal position x of the cursor.
CRSLIN	Return the vertical position y of the cursor.
LOCATE x,y,cursor	Place the cursor at (x,y) , e.g. for a following PRINT statement.

Graphics Mode	Purpose
GCLS	Clear the graphics screen.
COLOR fore,back,set	Select foreground and background color for the given color set.
PSET (x,y),index	Set the pixel at (x,y) with the given color [0...3].
PSET (x,y)	Set the pixel at (x,y) with the current foreground color.
PRESET (x,y)	Set the pixel at (x,y) with the current background color.
LINE (x1,y1)-(x2,y2),PSET	Draw a line from (x1,y1) to (x2,y2) with the foreground color.
LINE (x1,y1)-(x2,y2),PRESET	Draw a line from (x1,y1) to (x2,y2) with the background color.
POINT (x,y)	Return the color index of the pixel at (x,y) . [0...3, 10...13]

```

MH-20 Display - [80x24 chars] - [480x227 pixels]
EPSON MH-20
LIST
10 REM =====
20 REM
30 REM
40 REM =====
50 PRINT "SEQUENTIAL"
60 OPEN "O",#1,"A:SEQ.TXT"
70 FOR I=1 TO 64
80 PRINT#1, I
90 NEXT I
100 CLOSE #1
110 REM
120 PRINT "RANDOM"
130 OPEN "R",#1,"A:RAN1.TXT"
140 OPEN "R",#2,"A:RAN2.TXT"
150 FIELD #1:4 AS A$,4 AS B$,8 AS C$
160 FIELD #2:4 AS D$,4 AS E$,8 AS F$
170 FOR I=1 TO 10
180 LSET A$=MKI$(I)
190 LSET B$=MKS$(I)
Break

```

Figure 22: MH-20 in text mode after a **SCREEN 1,0** and a **LIST** command.

The caption bar shows the dimensions in character cells as well as in pixels.

Some differences from the Epson Specifications:

- Only a subset of the possible commands has been implemented. The program may handle unknown commands ungracefully.
- Text lines extending over multiple screen lines are not supported. Each line must fit on one line.
- In graphics mode, all dimensions have been doubled for better visibility – i.e. a line is drawn two pixels wide. The screen dimensions in pixels as shown in the title bar reflect this scaling and show the available coordinate space.
- The screen size can be considerably larger than that of the original display controller. Its size was limited to a text display of 16×32 characters respectively resolutions of 128×96 for monochrome graphics or 128×64 for color graphics.
- The size of the graphics screen is directly linked to the text screen size and cannot be changed. No movable window is implemented as this does not make too much sense on this larger screen.
- Both color sets of 4 colors each have been implemented as per specification. As they are only vaguely specified the default background color “green” has been made dark to have the default text color “yellow” stand out sufficiently. It is possible to use both color sets on the same screen, which was probably not possible on the original hardware.
- The `POINT` function returns 0...3 for colors in the color set 0, and 10...13 for colors from set 2. This allows distinguishing between the two color sets. The original hardware probably only returned values within 0...3.
- A context menu (right mouse button) allows copying the contents of the display to the clipboard. Depending on the current display mode, text and/or bitmap format are available.

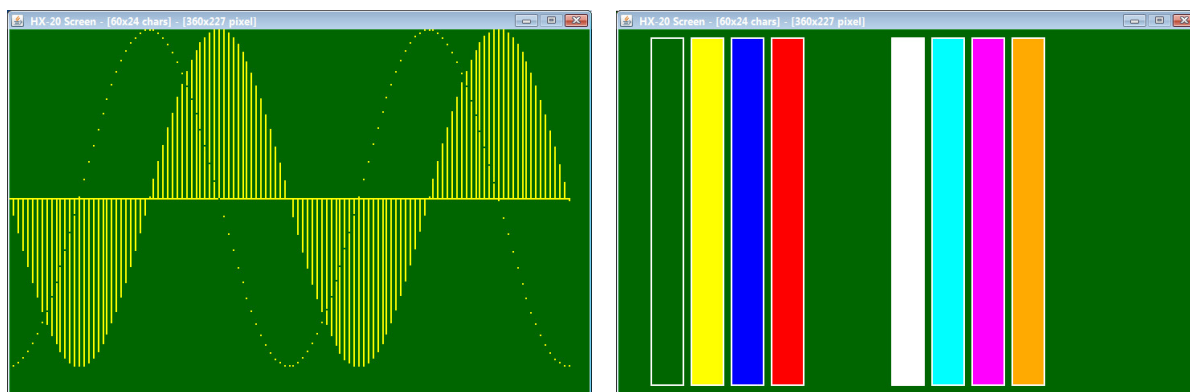


Figure 23: Result of running a simple plot programs.

Left: The same program runs on the internal LCD. For the external screen only a `SCREEN 0,1` command and individual scaling factors for the x- and y-direction have been added.

Right: The two color palettes (0 and 1) with 4 colors each, selected by using the `COLOR` command. The first bar (color index 0) represents the default background color of each color set.

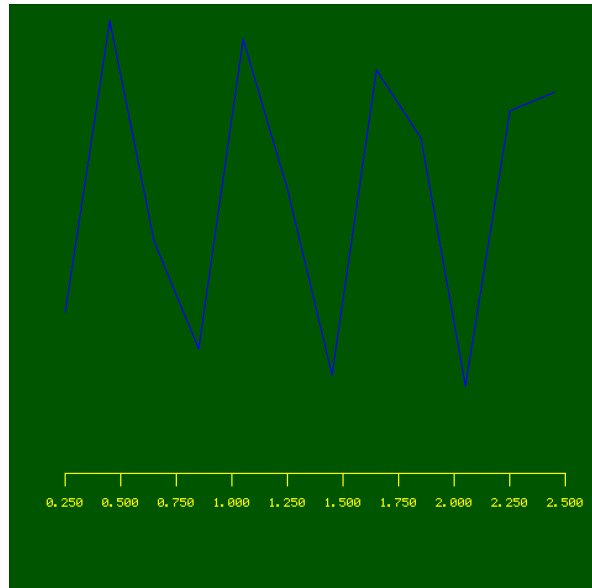


Figure 24: In contrast to the original Display Controller the software emulator can also display characters if a special machine language subroutine is used.

The example shown in Figure 24 uses a machine language subroutine to send a special data packet to the MH-20 Display Controller. The parameters of this subroutine are the X, and Y coordinates as well as the string to output (up to 32 characters). These are packed into a string because **USR** functions only allow for one parameter.

```

; a09 outchar.asm -loutchar.lst
; python LST2BAS.py outchar.lst > outchar.bas
OPT H01

                ORG      $0A40

BUFLEN EQU      32      ; max. string length
SERSND EQU      $FF70   ; operating system function
; BASIC floating point accumulator to return result
FPTYP EQU       $0085   ; 2 bytes: type of # in FPACC
FPACC EQU       $00D5   ; floating point accumulator

; Epson HX-20
; USR function for sending a string with leading
; 16-bit x-y coordinates via serial interface.
; The string may have up to BUFLEN characters.
; Returns the length of the output string
; (minus the 4 leading bytes)
;
; Usage:
; DEFUSR1=&H0A40
; DEFFNLO$(X%)=CHR$(X% AND 255)
; DEFFNHI$(X%)=CHR$((X%\256) AND 255)
; X=25 : Y=50
; M$=FNHI$(X)+FNLO$(X)+FNHI$(Y)+FNLO$(Y)
; L=USR1(M$+"Hello World")
;
; X points to string descriptor:
; 0,X: length of string, must be >4
; 1,X: address of string
0A40 8103          CMPA    #$03      ; do we have a string?
0A42 2653          BNE     OOPS      ; no: leave
0A44 E600          LDAB    0,X       ; length of string -> B
0A46 5A           DECB    0,X       ; minus 1 = data length
0A47 F70A9E        STAB    CNT      ; store data length

```

0A4A	C003		SUBB	#\$03	; minus X,Y
0A4C	2F49		BLE	OOPS	; less than one character?
0A4E	C120		CMPB	#BUFLEN	; up to BUFLen chars
0A50	2F02		BLE	LENOK	
0A52	C620		LDAB	#BUFLEN	; min(N,BUFLen)
					;
0A54	9602	LENOK:	LDAA	\$02	; return data type: integer
0A56	9785		STAA	FPTYP	; type of # in FPACC
0A58	4F		CLRA		; store integer in FPACC+2,3
0A59	97D7		STAA	FPACC+2	; high byte = 0
0A5B	D7D8		STAB	FPACC+3	; low byte = length
					;
0A5D	EE01		LDX	1,X	; address of string -> X
0A5F	A600		LDAA	0,X	
0A61	B70A9F		STAA	XPNT	; high byte of X
0A64	A601		LDAA	1,X	
0A66	B70AA0		STAA	XPNT+1	; low byte
0A69	A602		LDAA	2,X	
0A6B	B70AA1		STAA	YPNT	; high byte of Y
0A6E	A603		LDAA	3,X	
0A70	B70AA2		STAA	YPNT+1	; low byte
0A73	37		PSHB		
0A74	CC0AA3		LDD	#CHAR	; starting address of CHAR
0A77	FD0A98		STD	CPTR	; store pointer
0A7A	33		PULB		; length of string -> B
					; address of source char is (X+4)
					; address of destination is in CPTR
0A7B	A604	NEXT:	LDAA	4,X	; get next character A=*(X+4)
0A7D	3C		PSHX		; save source address
0A7E	FE0A98		LDX	CPTR	; destination address X=CPTR
0A81	A700		STAA	0,X	; store character code *CPTR=A
0A83	38		PULX		
0A84	08		INX		; increment source address
0A85	7C0A99		INC	CPTR+1	; increment low byte of target
0A88	2803		BVC	NOVER	; V=0: no overflow
0A8A	7C0A98		INC	CPTR	; else: increment high byte
0A8D	5A	NOVER	DECB		; decrement character count
0A8E	26EB		BNE	NEXT	; next character
0A90	4F		CLRA		; A=0: send a packet
0A91	CE0A9A		LDX	#PACKET	; address of PACKET
					;
0A94	BDFF70		JSR	SERSND	; send packet
0A97	39	OOPS:	RTS		
0A98	FFFF	CPTR:	FCB	\$FF,\$FF	; pointer to current CHAR
					;
					PACKET:
0A9A	00	OP:	FCB	\$00	; 0: send
0A9B	30	DID:	FCB	\$30	; destination ID
0A9C	20	SID:	FCB	\$20	; source ID
0A9D	EE	FCN:	FCB	\$EE	; my own function code
0A9E	03	CNT:	FCB	\$03	; data length - 1
					DATA:
0A9F	FFFF	XPNT:	FCB	\$FF,\$FF	; X
0AA1	FFFF	YPNT:	FCB	\$FF,\$FF	; Y
0AA3	FFFFFFFFFFFFFFF	CHAR:	FILL	\$FF,BUFLen	; buffer[BUFLen]
0AAA	FFFFFFFFFFFFFFF				
0AB1	FFFFFFFFFFFFFFF				
0AB8	FFFFFFFFFFFFFFF				
0ABF	FFFFFFFF				
					END

The corresponding BASIC loader and test program as created by the python script `LST2BAS.py` is:

```
10 REM --- Epson HX-20 ---
20 REM --- String Output to MH-20 ---
30 REM --- M. Hepperle 2018 ---
40 GOSUB 70
50 PRINT USR1(CHR$(0)+CHR$(32)+CHR$(0)+CHR$(64)+"Hello World")
60 STOP
70 REM --- Hex Code Loader ---
80 N%=0
90 MEMSET &HAC3
100 READ C$
110 IF C$="DONE" THEN 160
120 N%=N%+1 : IF N%=8 THEN PRINT ". "; : N%=0
130 C%=VAL("&H"+C$)
140 IF LEN(C$)=4 THEN A%=C% : GOTO 100
150 POKE A%,C% : A%=A%+1 : GOTO 100
160 PRINT
170 DEFUSR1=&H0A40
180 RETURN
190 DATA 0A40,81,03,26,53,E6,00,5A,F7,0A,9E,C0,03,2F,49,C1,20,2F,02
191 DATA C6,20,96,02,97,85,4F,97,D7,D7,D8,EE,01,A6,00,B7,0A,9F,A6,01
192 DATA B7,0A,A0,A6,02,B7,0A,A1,A6,03,B7,0A,A2,37,CC,0A,A3,FD,0A,98
193 DATA 33,A6,04,3C,FE,0A,98,A7,00,38,08,7C,0A,99,28,03,7C,0A,98,5A
194 DATA 26,EB,4F,CE,0A,9A,BD,FF,70,39,FF,FF,00,30,20,EE,03,FF,FF,FF
195 DATA FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF
196 DATA FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,DONE
```

18.4. Disk Drive Emulation

The second function of the MH-20 program is the emulation of disk drive units. This gives you four simulated floppy disk drives.

Note that a tooltip with a short directory listing is shown when you hover the mouse pointer over one of the drive images.

18.4.1. Technical Background

The Epson TF-20 dual 5-1/4" disk drive unit is actually a small computer which runs a variant of the CP/M operating system. It communicates with the HX-20 over a “high-speed” serial connection at 38400 baud using the EPSP Protocol developed by Epson. This protocol underwent some extensions for later Epson computers and is only sparingly documented.

When the HX-20 boots up, it first asks the TF-20 for a short boot loader program. After this has been received, it asks for a longer machine language program containing the code to extend the BASIC of the HX-20. This program implements the additional or modified keywords and commands to support the disk drive.

The extension code is loaded into the memory of the HX-20. Its actual location depends on the size of the RAM installed in the HX-20. Therefore, the HX-20 also asks the TF-20 to relocate the code according to its memory configuration. Thus, the TF-20 has to recalculate the affected addresses in the code before sending it to the HX-20. The MH-20 emulator supports all logical disk functions as required for operation of the HX-20.

18.4.2. The Emulation

The MH-20 emulator emulates two floppy units, i.e. a total of four disk drives. These are mapped to four directories:

DISK_A
DISK_B
DISK_C
DISK_D

Each directory contains individual files.

While the original floppy disks have a limited capacity, the capacity of the mapped drives is only limited by the mass storage capacity of the host computer. Of course it makes sense to limit the number of files in each directory to a reasonable number.

For this purpose each file is directly represented by a disk file on the host computer - no disk image files are used. Therefore, physical disk functions, like formatting and sector reading/writing, do not make much sense and produce no result.

The main applications of the disk emulation are

- saving and loading programs,
- creating, writing and, reading of data files.

18.4.3. Applicable BASIC Keywords and Commands

Keyword	Purpose
CLOSE	close file(s)
CVI, CVD, CVS	convert a string to numeric data
DSKF	return free space on disk (has no effect, always returns 320 KB)
DSKI\$	direct input of one record (has no effect, returns "Read Error")
DSKO\$	direct output of one record (has no effect, returns "Disk write protected")
EOF	return end of file code
FIELD	define fields for the record buffer used by random access file
FILES	display disk directory
FILNUM	define number of FCBs in advance
FRMAT	format a disk (has no effect)
GET	read one record from random access file
INPUT#	read data item from sequential access file
INPUT\$	read a string from a sequential access file
KILL	delete a file
LINE INPUT#	read line of characters from sequential access file
LIST	output a program listing to a file
LOAD	load a program from a file
LOADM	load a machine language program from a file
LOC	return the current record number of a file

LOF	return the largest record number of a file
LSET	store data in file buffer for random access file
MERGE	merge a program into current program
MKI\$, MKD\$, MKS\$	convert numeric data to a string
NAME	rename a file
OPEN	open a file
PRINT#	print data to a sequential access file
PRINT# USING	print formatted data to a sequential access file
PUT	write one random record from file
RESET	enable replacement of disk
RSET	store data in file buffer for random access file
RUN	load and execute a program from disk
SAVE	save a program to a file in binary or ASCII format
SAVEM	save memory range to a file
SYSGEN	create a new system disk (has no effect)
WHILE...WEND	conditional loop statement

Note that

- record numbers are 0-based
- each record is 128 bytes long
- the **FIELD** statement defines the structure of a complete record
- the **PUT** and **GET** statements write resp. read a complete record

18.5. Credits

Copyright notice for the serial library used in MH-20:

```

/* jSSC (Java Simple Serial Connector) - serial port communication library.
 * © Alexey Sokolov (scream3r), 2010-2014.
 *
 * This file is part of jSSC.
 *
 * jSSC is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * jSSC is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with jSSC. If not, see <http://www.gnu.org/licenses/>.
 *
 * If you use jSSC in public project you can inform me about this by e-mail,
 * of course if you want it.
 *
 * e-mail: scream3r.org@gmail.com
 * web-site: http://scream3r.org | http://code.google.com/p/java-simple-serial-connector/
 */

```

19. Map of the System RAM

The HX-20 system uses the lower part of its RAM for storage of various system variables. When writing assembler programs it is useful to have a complete picture of the RAM usage. The following Table was compiled from the Technical Manual.

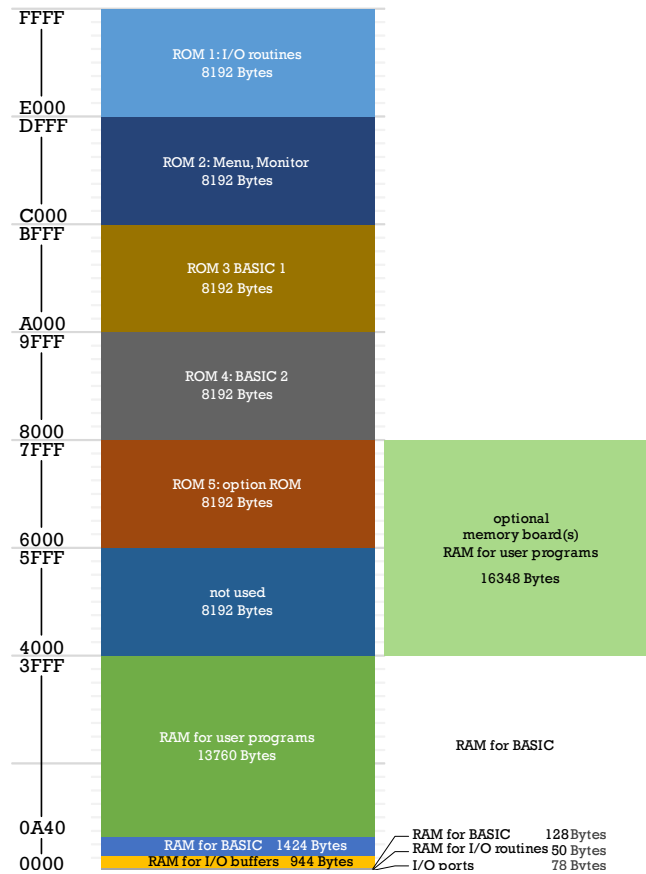


Figure 25: Global memory map of the HX-20 system.

Usage	Addr.	Len.	Name
Clock I/O ports	40	1	seconds
	41	1	alarm seconds
	42	1	minutes
	43	1	alarm minutes
	44	1	hours
	45	1	alarm hours
	46	1	day
	47	1	date
	48	1	month
	49	1	year
	4A	1	control register A
	4B	1	control register B
	4C	1	control register C
	4D	1	control register D
Clock	4E	50	RTC RAM
Basic interpreter	80	128	
Interrupt processing	100	3	INTCLK
	103	3	INTXT
	106	3	TRAP
	109	3	IRQ1 SCI
	10C	3	IRQ1 TOF
	10F	3	IRQ1 OCF
	112	3	IRQ1 ICF
	115	3	IRQ1 == INTCLK
	118	3	SW1
	11B	3	NMI

Usage	Addr.	Len.	Name
Vectors	11E	2	user defined chars.
	120	2	BRKADR
	122	2	MENADR MENU
	124	2	PAUADR BREAK
	126	2	CT3ADR PF3
	128	2	CT3ADR PF4
	12A	2	CT3ADR PF5
	12C	2	CT3ADR PF5rmbadr
	12E	2	PRMCNT
	130	2	WAKADR
	132	2	POFADR
	134	2	BSWTAD
	136	2	BSWBAD
	138	2	no name
Menu and link tables	13A	2	BITMP0
	13C	2	BITMP1
	13E	2	LNKTBL
Keyboard	140	1	KSTKSZ
	141	1	KICNT1
	142	1	KICNT2
	143	2	KICNTM
	145	10	NEWKTB
	14F	10	OLDKTB
	159	10	CHKKTB
	163	2	KYISAD
	165	1	KYISFL

Usage	Addr.	Len.	Name
	166	1	KYISCN
	167	1	KYISPN
	168	1	STKCNT
	169	1	KEYMOD
	16A	1	ONKFLG
	16B	1	KPRFLG
	16C	1	KEYRPT
	16D	2	CKEYRD
	16F	18	KYISTK
	181	8	CHRSTK
	189	7	no name
Microprinter	190	6	CHRPTN
	196	1	COLCNT
	197	24	CHRDAT
RS 232C	1AF	2	RSBAUD
	1B1	2	RSCRC
	1B3	2	RSBCC
	1B5	1	RSBITL
	1B6	1	RSMODS
	1B7	1	RSSTSR
	1B8	2	RSBFAD
	1BA	2	RSBFBT
	1BC	2	RSBFSZ
	1BE	2	RSINP
	1C0	2	RSOUP
	1C2	2	RSDCNT
High Speed Serial	1C4	1	SRFMT
	1C5	1	SRDDEV
	1C6	1	SRSDEV
	1C7	1	SRFNC
	1C8	1	SRSIZ
	1C9	1	SRACKC
	1CA	1	SRTRCN
	1CB	1	SRTIMO
	1CC	1	SRETMO
	1CD	1	SRATMO
	1CE	1	SRMODE
	1CF	1	STETDL
	1D0	1	SRBLCN
	1D1	1	SRERMD
	1D2	1	SRRVFL
	1D3	2	SREIX
External Cassette	1D5	1	CSMOD
	1D6	2	CSBLNO
	1D8	2	CSBCC
	1DA	2	CSBLSZ
	1DC	1	CSSTP
	1DD	1	CSSTS
	1DE	2	CSBFAD
	1E0	2	CSBFRT
	1E2	2	CSBFSZ
	1E4	2	CSBFIP
	1E6	2	CSBFOP
	1E8	2	CSBFCM
	1EA	1	CSRDTR
	1EB	1	CSRDCN
Internal Cassette	1EC	1	MSMOD
	1ED	2	MSGLNO
	1EF	2	MSBCC
	1F1	2	MSBLSZ
	1F3	1	MSBSTP
	1F4	1	MSSTS
	1F5	2	MSBFAD
	1F7	2	MSBFBT
	1F9	2	MSBFSZ
	1FB	2	MSBFIP
	1FD	2	MSBFOP
	1FF	2	MSBFCM
	201	1	MSRDIR
	202	1	MSRDCN
	203	2	MSCNTR

Usage	Addr.	Len.	Name
	205	1	MSMNCM
	206	1	MSTOFCN
	207	1	MSPLMD
ROM Cartridge	208	1	PRMSTS
	209	2	STAPRS
	20B	2	FTADRS
	20D	2	EDADRS
Binary Dump/Load	20F	2	DLTPAD
	211	2	DLBTAD
	213	2	DLOPAD
	215	2	DLSTAD
	217	1	DLDVID
	218	1	DLSTS
	219	2	DLDVIX
RESERVED	21B	5	reserved
LCD	220	80	PSBUF
	270	2	SCRTOP
	272	2	SCRBOT
	274	2	DISTOP
	276	1	VSCRX
	277	1	VSCRY
	278	1	CURX
	279	1	CURY
	27A	1	LRMODE
	27B	1	UDMOD
	27C	1	CURMRG
	27D	1	SSPEED
	27E	1	DISPX
	27F	1	DISPY
	280	1	DISSTS
	281	5	no name
	286	6	CHRPTN
Screen work area	28C	20	no name
Monitor work area	2A0	48	no name
External Cass. Headers	2D0	1	CHBLID
	2D1	2	CHBLNO
	2D3	1	CHBLBU
	2D4	4	CID
	2D8	8	CFNAME
	2E0	8	CFTYPE
	2E8	1	CRTYPE
	2E9	1	CBMODE
	2EA	5	CBLNG
	2EF	5	no name
	2F4	6	CDATE
	2FA	6	CTIME
	300	6	no name
	306	2	CVOLN
	308	8	CSYSN
	310	20	no name
Internal Cass. Headers	324	1	MHBLID
	325	2	MHBLNO
	327	1	MHBLBU
	328	4	MID
	32C	8	MFNAME
	334	8	MFTYPE
	33C	1	MRTYPE
	33D	1	MBMODE
	33E	5	MBLNG
	343	5	no name
	348	6	MDATE
	34E	6	MTIME
	354	6	no name
	35A	2	MVOLN
	35C	8	CSYSN
	364	20	no name
	378	260	CASBUF
Total RAM used		1148	bytes

Table 2: Detailed RAM usage by the HX-20 operating system.

20. News and Commercial Announcements

Note: The following figures contain company and product names which are reproduced here only for historic documentation and archival purposes. These companies may not exist anymore and the products mentioned are surely not available anymore.

Actual size.

The size, of course, is a dead giveaway. But don't let the size fool you. The HX-20 is not a toy. Or a glorified calculator.

It's a computer.

A real computer, with 16K RAM (optionally expandable to 32K), and 32K ROM (optionally expandable to 64K). RS-232C and serial interfaces, a full-size ASCII keyboard, a built-in printer, a scrollable LCD screen, and sound generation. A micro-cassette and ROM cartridge are available as options.

Viva la difference!

In fact, the only difference between the Epson HX-20 and an ordinary computer are:

- 1) The HX-20 is small enough to fit inside your briefcase.
- 2) It'll run on its own internal power supply for 50-plus hours, and fully recharge in less than eight.
- 3) It gives you up to 50 program functions at the push of a button.
- 4) It lets you interface with peripherals like the MX Series printers for correspondence quality output, the CX-20 Accuwrite Computer for remote communications, a barcode reader for inventory control, and an audio cassette for loading and saving programs.
- 5) It lets you shut the whole unit off while preserving all programs in RAM, and, just, but far from here.
- 6) It costs less than \$300. That's right—less.

The perfect traveling companion.

With the Epson HX-20 and the optional RAM expansion, you'll be able to compute just about anywhere. Because it includes custom-made batteries and a low-power, all-CMOS memory keep the HX-20 running for over 50 hours. And even if you shut the HX-20 off, a low-voltage system maintains all programs you have in RAM.

Little screen, big picture.

The HX-20's unique scrollable LCD screen is the ultimate answer to the question, "How do you get a big screen in a small space?" You just show part of it at a time.

Option: Measurements

EPSON HX-20

*** MENU ***
CTRL Initialize
1 MONITOR
2 BASIC

PRINTER OFF ON PAPER FULL

PAUSE MENU BREAK PF1 PF2 PF3 PF4 PF5

NUM HOME CLR SCRN INS DEL

! " # \$ % & ' () _ = { } \ |
TAB Q W E R T Y U I O P ^ @
CTRL A S D F G H J K L ; : *
SHIFT Z X C V B N M , > ? / SHIFT GRPH

CHG LOCK

MICRO CASSETTE DRIVE REC

So with the HX-20, you can do programming, word processing and data entry just like you've got a big screen, up to 255 characters wide, with easy-to-read upper and lower case letters, numbers and punctuation and any 20 column by four line part of it visible by user command. Built in hardcopies.

The HX-20's built-in 24 column dot matrix impact microprinter hands hardcopies to you at 42 LPM, in a crisp, precise 5x7 matrix. It even has bit addressable graphics to give you a print-sized sales chart, and enough international symbols to print most Western languages.

Epson makes more and better printers than anyone else in the world. Need we say more?

The best is yet to come.

When you hold an HX-20 in your hand, you're not only holding a lot of capacity, you're holding a lot of expansion.

There's a standard cassette interface, a cartridge interface, the RS-232C and serial interfaces, and a system bus that lets you expand RAM and ROM capabilities. There's even a floppy disc drive for mass capacity in a mini package.

The Epson edge.

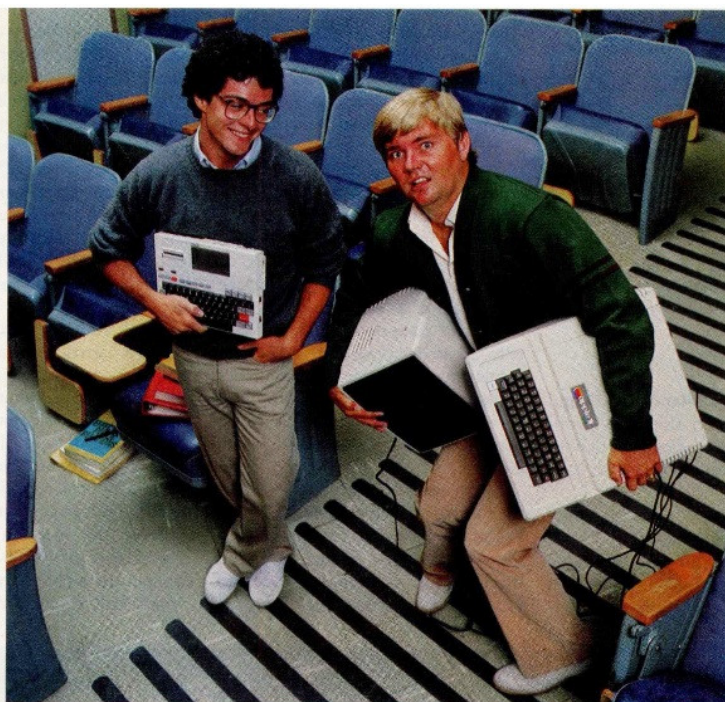
Surprised that a computer like the HX-20 should come from Epson? You shouldn't be. Because we've been building computers in Japan since 1976. And we've been practicing ultra-high-quality precision manufacturing for a lot longer than that.

We didn't jump right into the American microcomputer market. We could afford to take our time to wait for the product that was going to stand America on its ear.

This is it.

The Epson HX-20.

EPSON
EPSON AMERICA, INC.
COMPUTER PRODUCTS DIVISION
3415 Kashiwa Street
Torrance, California 90505
(213) 539-9140



Which do you think is the more sophisticated computer?

Epson.

The big differences between the Epson HX-20 Notebook Computer (on the left) and the Apple Computer (on the right) are: 1) the HX-20 doesn't need a power cord, 2) the HX-20 weighs only about four pounds, and 3) the HX-20 costs a lot less money.

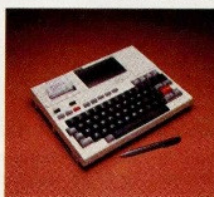
The Epson HX-20 Notebook Computer has a full-size keyboard, a built-in LCD screen, a built-in printer, 48K of combined RAM and ROM memory, and an internal power supply that will keep it running for over 50 hours. So you can do computing and word processing virtually anywhere you happen to be. Whereas, with the Apple Computer, you can only go as far as an extension cord will take you.

And on the HX-20, you get communications interfaces, upper and lower case letters, five program areas, a full 68 keys including an integrated numeric key pad, an internal clock/calendar, and the screen and printer. Standard. On the Apple, you pay something extra for each feature — if you

can get them at all.

All of which makes the take-it-anywhere HX-20 perfect for business executives, salespeople, students, kids — anyone who's looking for an affordable, practical way into computing.

Portable. Powerful. Affordable. Sophisticated. The extraordinary HX-20 Notebook Computer. Find out just how extraordinary. Call (800) 421-5426, in California (213) 539-9140 for your nearest Epson computer dealer.



EPSON
EPSON AMERICA, INC.

Circle 177 on inquiry card.

BYTE March 1983 99

ROSE BOWL SCOREBOARD SNAFU DONE WITH PORTABLE COMPUTER

During January's Rose Bowl, a scoreboard prank by two CalTech students was made possible by two computers and radio modems. The students, who are now being prosecuted for trespassing, used an Epson HX-20 notebook-size portable computer with an RF modem to tap into an 8086 breadboard they'd attached between the scoreboard and its operators. The students put several messages on the scoreboard's scratch-pad area and finally changed the names of the teams to show CalTech trouncing rival MIT, instead of UCLA beating Illinois. The students later held a seminar called "Packet RF Control of Remote Digital Displays."

BYTE April 1984 9

EPSON

Hand-Held-Computer HX-20
Der Computer, der mit auf die Reise geht.



Netzunabhängig.
DIN A 4 groß.
Leicht. 1.600 Gramm.
max. RAM 32 KB. ROM 40 KB.
MICROSOFT-BASIC.
LCD-Display.
Mini-Drucker mit Graphikfähigkeit.
Schreibmaschinentastatur.

EPSON

Epson-Technologie. Der Erfolg besserer Ideen.
Epson Deutschland GmbH · Am Seestern 24 · 4000 Düsseldorf · Telefon: (0211) 5961001

Figure 26: Can these eyes lie? The dark color scheme of the HX-20 seems to have been composed specifically for this advertisement.

**Never before has so much been available
from so little, to so many.**



Never before in the history of modern computers has there been available a totally portable machine with the ability to cope with so many demands, to so many people, in so many different business areas.

Epson, with over 20 years experience in designing and manufacturing high quality printers, have produced the HX-20, a precision machine with its own rechargeable power supply that can be used for just about any task within today's discerning business: from data capture to word processing, from card indexing to sales order entry. Communicating with other machines is no problem and the HX-20 is easily coupled to one of our fine printers. You can even link in another main computer system by using an acoustic coupler.

Don't be fooled by its size, the HX-20 has all the software back-up you'd expect from a much larger machine and incorporates many "bigger computer" features - 16k RAM expandable to 32k with serial interfaces, a full size typewriter keyboard, its own built in

LCD screen and a dot matrix microprinter. A micro-cassette facility is available as an optional extra.

A complete computer that will either stand on its own or could be the obvious extension to your existing system.

More and more people are finding out just how big the small compact HX-20 is. Why don't you find out for yourself - you owe it to your business.

EPSON

**Extraordinary product.
Exceptional quality.**

Epson (UK) Limited, Freeport,
Wembley, Middlesex HA9 6BR.
Sales Enquiries: Freefone EPSON.
General Enquiries: 01-902 8892.
Telex: 8814169.

- ☐ I would like a demonstration of the HX-20 Portable Computer.
☐ Please send me details of my local stockist.

Name

Position

Company

Address

Tel:

PC2/20

● Circle No. 106

Figure 27: This ad in the February 1982 issue of "Practical Computing" obviously aims at British customers, following Winston Churchill's famous words.



Try this with an ordinary computer.

Epson.

The new Epson HX-20 is no ordinary computer. Not by a long shot. It's the world's only Notebook Computer with the power of a desktop and the portability of a handheld.

So you can do serious computing, data processing, even word processing. Anytime. Anywhere.

To start with the HX-20 has 16K RAM (optionally expandable to 32K), 32K ROM (optionally expandable to 64K), RS-232C and serial interfaces, a full-size ASCII keyboard, a built-in micro-printer with dot addressable graphics, a scrollable LCD screen, five programmable function keys, and . . . well, that's just the beginning.

The HX-20 is small enough to tuck inside a briefcase or under your arm. It runs on internal power for 50-plus hours and recharges in eight. It lets you interface with peripherals like MX Series printers, the CX-20 battery-powered acoustic coupler, a barcode reader, and audio cassette. And you can even get it with options like a micro-

cassette drive, ROM cartridge, floppy disk and display controller.

Now, prepare to have your mind boggled by one more feature: the price. The Epson HX-20 Portable Notebook Computer retails for less than \$800. That's right — less. Which means it's just right for students, businesspeople, kids — anybody who's looking for an affordable way into serious computing.

Powerful. Portable. Affordable. The HX-20 is just what you'd expect from Epson.
The extraordinary.



EPSON
EPSON AMERICA, INC.
COMPUTER PRODUCTS DIVISION

3415 Kashiwa Street • Torrance, California 90505 • (213) 539-9140

Circle no. 138 on reader service card.

Figure 28: And another extraordinary one lifted from the October 1982 issue of "Dr. Dobbs Journal".



The new HX-20 Notebook Computer.
Where was it when I was a kid?

Epson.

The new Epson HX-20 Notebook Computer is perfect for kids. But it's not just for kids.

The HX-20 has as much total memory as most popular desktop computers. And, like a desktop, you can connect it to a monitor, add extra memory, use a cassette or microcassette to load and store programs, play games, even interface with other computers through a telephone modem.

But that's where the similarity ends. Because, unlike a desktop, the HX-20 has the hardware you need to do word and data processing anywhere. Built in. It has enough internal power to run for 50-plus hours, a full-size keyboard, a scrollable LCD screen, even a handy little microprinter.

But more importantly, it has something that no comparable personal computer can match: a price tag of under \$800. You can have *everything* you need to do real computing for a lot less than the cost of most desktop computers. A lot less.

The new Epson HX-20 Notebook Computer. It's perfect for kids, salespeople, business executives, students — anyone who's looking for an affordable way into serious computing.

For the Epson computer dealer nearest you, just call (800) 421-5426, or in California (213) 539-9140.

Try it out. After all, why should kids have all the fun?



EPSON
EPSON AMERICA, INC.
COMPUTER PRODUCTS DIVISION

3415 Kashiwa Street • Torrance, California • (213) 539-9140

CIRCLE NO. 25 ON FREE INFORMATION CARD

Figure 29: This ad was taken from the March 1983 issue of "CE".

HX 20 schon 'geknackt'

(Leserbrief von K. H. Kreeb, Worpssede, in c't 5/84)

Die 'interne Software' des HX-20, für die sich Herr Kreeb interessiert, ist schon seit längerer Zeit geknackt. Wir sind drei HX-Freaks und geben seit Sommer 1983 eine HX-20-Fachzeitschrift 'EPSILON' heraus. Diese erscheint 6 mal pro Jahr und wird momentan von über 400 Personen in ganz Europa abonniert. Daneben vertreiben wir eine HX-20-Dokumentation, die die Betriebsroutinen und Systempointer des HX erläutert und auflistet, ein sehr leistungsstarkes Textverarbeitungsprogramm, ein Debugger/Compactorprogramm und ein EPROM-Programmiergerät. Am 3. März 1984 veranstaltete EPSILON eine HX-Tagung, an der 60 Abonnenten teilnahmen, u. a. auch aus der Bundesrepublik und aus Österreich. Am 27. Oktober 1984 findet die zweite Tagung statt, die unter dem Generalthema 'Kommunikation mit EPSON-Computern' stehen wird.

Gerne senden wir Herrn Kreeb und allen, die sich interessieren, eine Probenummer zu.

Peter Addor, EPSILON,
Postfach 185,
CH-8704 Herrliberg-Zürich

c't 1984, Heft 6

Und wieder einmal 'HX-20 geknackt'

(Leserbriefe c't 5, 6, 7/84)

Mit Interesse habe ich die Briefe zum Thema 'HX-20' verfolgt. Daraus läßt sich schließen, daß wahrscheinlich die miserable Dokumentation der meisten Computerhersteller schon zum Standard erklärt wird. Beim HX-20 gibt es jedoch nicht mehr so viel zu knacken, da in dem von EPSON vertriebenen 'Technical Reference Manual' das Monitorprogramm mit vielen Beispielprogrammen in Assembler erklärt ist. Sicher bleiben noch einige Geheimnisse zu lüften, jedoch liegt der Verdacht nahe, daß auch User-Clubs ohne Erwähnung der Quelle aus diesem Fundus zehren. Allein der Preis von DM 300,— trübt die Freude an diesem ansonsten vorbildlichen Werk.

Knut Brenndörfer, Ismaning

c't 1984, Heft 8

Figure 30: The quest for finding more technical information shows up in letters to the German magazine c't.

NEU · HX-20 FORTH · NEU
Die neue Programmiersprache:
Durch einfaches Einstecken des FORTH-EPROM's in den freien Stecksocket im HX-20, haben Sie zusätzlich zu BASIC die Programmiersprache FORTH zur Verfügung.
Es wird kein Speicherplatz belegt (RAM bleibt frei).
FORTH-EPROM DM 198,-
FORTH Handbuch, ca. 100 Seiten, englisch DM 45,-, ab Nov. '83 deutsch DM 79,-
Preis inklusive Mehrwertsteuer
Weitere Programme: Adressverwaltung, Statistik, Kalkulationen, Datenbank, Kreditkalkulationen, Sendefunktion usw. Programmiersysteme nach Ihren speziellen Erfordernissen. Fordern Sie unsere Programmübersicht an!
time-soft-edu
Sophienstraße 32 · 7000 Stuttgart 1 · Tel. 0711/22 84 71/72
Programme + Computer für zeitgemäße Anwendungen

c't 1984, Heft 2

NEU HX-20 & CP/M
HX-20-Video-Adapter jetzt
die komfortable Verbindung zum Monitor!
STOP
HX-20-Floppy-Set (bis 1,2 MB)
1-2 Laufwerke, je 320-640 K, voller HX-20-Befehlssatz, Video-Adapter und Floppy in gleichem oder separatem Gehäuse. CP/M*-Betriebssystem, zusätzlich CP/M*-Programme einsetzbar.
CP/M ist ein Warenzeichen der Digital Research, Inc.
time-soft-edu
Sophienstraße 32 · 7000 Stuttgart 1 · Telefon: 0711/22 84 71/72
Programme + Computer für zeitgemäße Anwendungen

c't 1984, Heft 3

Deutschlands größter HX-20-Anbieter!!!
HX-Super-Video-Adapter V-2 (auch M-10 + TANDY)
7x10-Punkt-Matrix, gestochen scharfe Anzeige mit Unterlängen.
Darstellung: 80 Zeichen x 24 Zeilen und 2 Statuszeilen (z.B. Zeile, alternativ) ein- und ausblendbar, sämtliche Steuerbefehle - umschaltbar per Programm oder Tastatur (ESCAPE-SEQUENZEN). Kompletter HX-20-Zeichensatz (inkl. aller HX-20-Grafikzeichen). Zusatzumschaltung auf 40 x 24, 40 x 12 und 20 x 8 - mit entsprechend vergrößerter Darstellung auf dem Monitor. Anzeige:
● stehend, blinkend und invers (auch gemischt möglich).
Kleines formschönes Gehäuse (145/200/80 ca.). Sofort lieferbar. DM 698,- inkl. weitgehend wie V-2, jedoch hochauflösende Grafik mit 512 x 512 Punkten - einzeln setzbar/löschar. Ab 4/84
NEU HX-20-3,5" VIDEO-DISC DM 3398,- inkl.
einschließlich eingebautem Video-Adapter
HX-20-Mikro-Floppy-Set 3,5" (wahlweise auch 5 1/4") bis 1,5 MB (Mega-Byte) 1-3 Laufwerke, je 380-760 KB, voller HX-20-Befehlssatz, mit integriertem Video-Adapter (V-1)*, CP/M*-Betriebssystem durch Z80-CPU in der Floppy, 64 K - Hauptspeicher - HX-20 als Keyboard - Durch CP/M haben Sie Zugriff auf eine der größten Software-Bibliotheken...
Achtung: HX-20 Video-Adapter/Floppy-Set haben Original-EPSON-HIGH-SPEED-SERIAL-Anschluss. V24/RS232C Schnittstelle bleibt frei für Drucker/Modem usw. Bitte vergleichen Sie!
● Gesamtprogramm gegen 1.30 DM in Briefmarken!
Programme + Computer für zeitgemäße Anwendung.
time-soft-edu
Sophienstr. 32
7000 Stuttgart 1
Tel. (0711) 22 00 71
Telefax: 7 22 706 lss d

c't 1984, Heft 9

Figure 31: The company "time-soft" had many special offers for HX-20 owners.

EPSON Manager-Computer

ARRO



Mit dem im Koffer steckt Sie keiner in die Tasche.
Weil Sie damit einen Assistenten an der Hand haben, der für Sie merkt, rechnet, kalkuliert, plant und schreibt.
Auf einer Fläche nicht größer als DIN A4.
Mit einem einzigartigen Multi-Programm, zu einem günstigen Preis.
Kommen Sie vorbei, testen Sie Ihren neuen Mitarbeiter.

mirwald electronic
Fasanenstraße 8b, 8025 Unterhaching/München,
Telefon (0 89) 6 11 12 24, FS 5 213 476
Büro Frankfurt: Adalbertstr. 15
Telefon (06 11) 70 35 38

Technologie, die Zeichen setzt.

c't 1984, Heft 9

Neu !

HX20 - Micro Terminal

DM 1298,- inkl. MwSt.



Dieses neue MICRO - TERMINAL für den EPSON HX20 Hand - Held - Computer gestattet die Darstellung von bis zu 80 Zeichen auf 25 Zeilen. Das 2000 Zeichen - Display mit grünem Schirmbild und Antireflexscheibe gewährleistet größtmögliche Benutzerfreundlichkeit. Sowohl Text, wie auch Graphik werden mit hoher Schärfe dargestellt. Eine hervorragende ergonomische Konstruktion gibt die Möglichkeit durch Drehen oder Kippen, das Sichtgerät auf optimalen Betrachtungswinkel einzustellen.

mirwald electronic BMC

Fasanenstraße 8b, 8025 Unterhaching/München,
Telefon (0 89) 6 11 12 24, FS 5 213 476
Büro Frankfurt: Adalbertstr. 15
Telefon (06 11) 70 35 38

Figure 32: Besides a display controller, Mirwald also sold memory expansion boards for the HX-20.

Handbuch sowie eine menügesteuerte Software, die das Programmieren der ICs im Standard- oder Intelligent-Modus erlaubt. Das Programmiergerät HXP2000 kostet als Bausatz 440 DM, als Fertigergerät 560 DM.

SES-Electronic, Im Grund 17, 6920 Sinsheim, 072 61/32 64.

HX-20 programmiert EPROMs

Das netzunabhängige Programmiergerät HXP2000 kann in Verbindung mit dem Epson-Computer HX20 alle gängigen EPROM-Typen programmieren. Zum Lieferumfang des Gerätes gehören ein (deutsches)



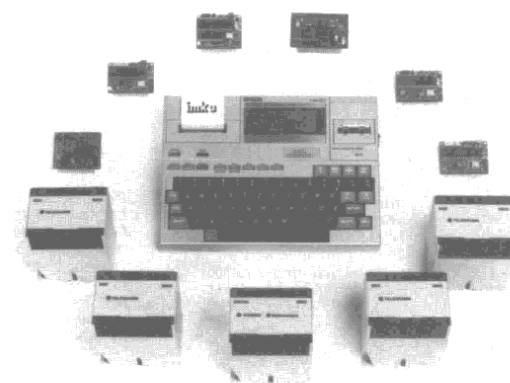
c't 1985, Heft 8

Daten erfassen

Die intelligenten Unterstationen der Serie IMP232 erlauben die Erfassung von analogen und digitalen Daten 'vor Ort'. Diese Daten können dann direkt über jede RS232-Schnitt-

stelle in einen Rechner gelesen werden. Über eine 'Kopfstation' können bis zu 32 Unterstationen dezentral an einer optoelektronischen Leitung angeschlossen werden.

Imko GmbH, Tulpenstraße 11, 7505 Ettlingen 5, 072 43/998 04.



c't 1985, Heft 9

Figure 33: More accessories like EPROM programmer and data acquisition systems were available from 3rd parties.

Mobiles Messen

Um Meßwerte vor Ort zu erfassen und diese vom PC auswerten zu lassen, bietet CSM eine preiswerte Lösung an. Mit dem ADDA-K in Verbindung mit dem Epson HX-20 hat der Benutzer ein hochwertiges mobiles Meßwerteerfassungs- und Verarbeitungssystem zur Verfügung. Die Adda-K-Schnittstelle hat acht über

Multiplexer anwählbare 12-Bit-Analogeingänge, je acht digitale Ein- und Ausgänge und ist direkt mit dem Systembus des HX-20 verbunden. Die serielle Schnittstelle des HX-20 bleibt bei dieser Lösung für andere Aufgaben frei. Der batteriegepufferte Meßwertspeicher ist von 32 KByte auf 128 KByte aufrüstbar. Das HX-20-

Basic ist um einige Echtzeitbefehle erweitert worden, mit denen z. B. der Zeittakt für die A/D-Umsetzung und die Speicherrate jedes Analogeingangs programmiert werden können. Mit dem mitgelieferten Plott-Programm werden die aufgezeichneten Meßwerte auf dem im HX-20 einge-

bauten Minidrucker bereits vor Ort ausgegeben. Die ADDA-K-Schnittstelle kostet mit 64 KByte Meßwertspeicher knapp 3300 DM.

CSM
Mühlwiesenstr. 1
7024 Filderstadt-Sielmingen
☎ 0 71 58/6 48 47



So handlich ist die Kombination ADDA-K und Epson HX-20

me 1/1987

Figure 34: And, another 12-bit A/D data acquisition system with digital I/O ports from CSM.



GRAPHIK-VIDEO-ADAPTER „EVA 1a“
ergänzt Ihren EPSON HX-20 zum komfortablen Computer-Arbeitsplatz.

Bis heute ist der „EVA 1a“ wegen seiner **Vollgraphik** von 480 x 200 Punkten und der vielen Extras schon über 3000mal im Einsatz – in mehr als 10 Ländern.

Technische Kurzdaten
Text-Modus: – 32, 40, 72 und 80 Z. bei 24 Zeilen
– Inverse Darstellung
– Hellere Darstellung
– DIN- und ASCII-Zeichensatz

Graphik: – 480 x 200 Punkte
– Testbild abrufbar
– Beschriftung mit 80 Z./Zeile
– Autom. Parameter-Begrenzung

Unsere Vertragshändler zeigen Ihnen mehr:

PLZ	Ort	Firma	Telefon
1000	Berlin 30	TCV-Stroetmann	0 30/2 62 14 85
1000	Berlin 31	Computer-Centrum CSM-Minhoff & Co. GmbH	0 30/8 54 50 55
2000	Hamburg	Walter Kluxen GmbH	0 40/23 70 10
2800	Bremen	Weber-Funk GmbH	04 21/49 00 19
3000	Hannover	TCV-Stroetmann	05 11/1 46 58
3000	Hannover	Computer-Centrum trendDATA Computer GmbH	05 11/1 46 29
4000	Düsseldorf	datec Software GmbH	02 11/48 20 85
4000	Düsseldorf	Helmut Rennen GmbH	02 11/30 60 98
4050	Mönchengladbach	Computer Commerce	0 21 61/1 87 64
4100	Duisburg	Helmut Rennen GmbH	02 03/2 49 26
4300	Essen	Helmut Rennen GmbH	02 01/23 71 30
4350	Recklinghausen	Computer Centrale	0 23 61/4 57 08
4400	Münster	Computer Stümpel	02 51/79 69 29
4600	Dortmund	Richard Müller GmbH	02 31/52 87 22
4630	Bochum	BO-DATA GmbH & Co. KG	02 34/70 22 77
5000	Köln	Büromaschinen Braun	02 21/21 91 71
5860	Iserlohn	Möller Bürotechnik	0 23 71/4 00 22
6000	Frankfurt	Kegelmann Computerhaus	0 69/44 60 16
6000	Frankfurt	M.A.S. GmbH	0 69/67 60 14
6100	Darmstadt	EBG Darmstadt GmbH	0 61 51/31 38 90
6200	Wiesbaden	MicroComp GmbH	0 61 21/4 53 77
6500	Mainz	Elphotec GmbH	0 61 31/23 19 47
6600	Saarbrücken	WIKO GmbH	06 81/6 34 44
6800	Mannheim	Josef Arz KG	06 21/2 38 44
7000	Stuttgart	Fritz-Computer GmbH	07 11/7 80 02 38
7800	Freiburg	Meisterknecht GmbH	07 61/3 16 66
7910	Neu-Ulm	Kumatronik GmbH	07 31/72 29 99
7980	Ravensburg	Kumatronik GmbH	07 51/95 41/43
8025	Unterhaching	Minwald Electronic GmbH	0 89/6 11 20 40
8500	Nürnberg	Wagner GmbH	09 11/32 90 60
8672	Selb	ROTRONIK P. Rogler	0 92 87/30 99

KK-SYSTEMS GmbH
Postfach 1248, 2803 Weyhe 1.

mc 12/1984

Figure 35: Another video controller was developed by KK Systems in the north of Germany.

RAM-Disk für HX-20

Eine Speicherkapazität von 2 x 128 KByte (netto) bietet die RAM-Disk RDSK1 für den Epson-Rechner HX-20. Die Disk ist in einem separaten Gehäuse untergebracht und schaltet sich bei Datentransfer automatisch ein, wodurch die Batterie-Kapazität mindestens für 12 Stunden Betrieb ausreicht. Die Disk erlaubt den Zugriff auf Daten mit 12facher Geschwindigkeit gegenüber dem normalen Diskettenlaufwerk und kostet etwa 1600 DM.

KK-Systems GmbH, Eichenstraße 5, 2808 Syke 2, 0 42 42/79 31.

c't 1985, Heft 12

CMOS RAM-DISK FÜR DEN HX-20

RDSK1, 2 x 128 KByte netto **DM 1 596,-**
RDSK2, 2 x 256 KByte netto **DM 2 536,-**

KK-SYSTEMS GmbH
Eichenstraße 5, 2808 Syke 2, Tel. 0 42 42/79 31

- Buchgröße 40 x 120 x 185 mm
- Gewicht nur 550 Gramm
- 2 x 120 bzw. 2 x 240 Directory-Einträge
- Datenerhalt bis zu 3 Monaten ohne Laden
- Ladeautomatik für NiCd-Akkus eingebaut
- Power-Down-Automatik für Datenschutz
- Bis zu 200mal schneller als Kassette
- Disk-BASIC voll TF-20-kompatibel
- 12 Monate Werks-Garantie

Unterlagen kostenlos anfordern

c't 1986, Heft 5

Figure 36: Also available from KK Systems: RAM disks for the HX-20.



Figure 37: Obviously, there were other disk drives available besides the Epson TF-20

21. References and Further Reading

- [1] Epson HX-20 - Technical Manual – Hardware.
- [2] Epson HX-20 - Technical Manual – Software.
- [3] Eratosthenes Sieve Benchmark Program, BYTE 1/1983.
- [4] E. Balkan, "Using and programming the Epson HX-20", Van Nostrand Reinhold, 1985.
- [5] <http://electricker.yxs4all.nl/comp/hx20/>
- [6] Brenndörfer, Knut, "Mehr Speicher für den HX-20", Magazin "mc" 4/1984, pp. 119-121.
- [7] Jebautzke, Michael, "Drucker am High-Speed Interface", Magazin "mc" 7/1985, pp. 82-83.
- [8] Bahmann, Wolfram, "Disassembler für HX-20", Magazin "mc" 7/1983, pp. 66-67.
- [9] Rohlf, Kristen, "HX-20 plottet Funktionen", Magazin "mc" 1/1984, pp. 86-87.
- [10] Gründler, Rolf, "Datenbank-Dialog mit dem HX-20", Magazin "mc" 12/1983, pp. 56-58.
- [11] Schnieder, Hermann, "HX-20 als Terminal", Magazin "mc" 2/1984, pp. 58-60.
- [12] Wald, Elizabeth, "Slipping Sideways", PCN February 1984.