the practical introduction to a powerful system

JUNIOR COMPUTER

Book 3

Elektor

The Elektor Junior Computer

the practical introduction to a powerful system

A. Nachtmann
G.H. Nachbar

Elektor Publishers Ltd.

The circuits published are for domestic use only. Patent protection may exist with respect to circuits, devices, components etc. described in this publication. The publishers do not accept responsibility for failing to identify such patent or other protection.

Printed in the Netherlands
ISBN 0905705 09 2

Junior is growing!

With the addition of the interface board, the Junior Computer can be transformed into a complete personal computer system. This board will form an essential link between the computer and the outside world as it includes additional RAM and EPROM and extra input/output possibilities. It also enables further memory cards and one or two cassette recorders to be added to the system. Thus, the Junior Computer is able to communicate with the operator (and vice versa) in a much more sophisticated manner.

Chapter 10 deals with the practical aspects of hardware involved in the extension. All constructional details are given together with modifications required to the main computer board and the power supply. It will be apparent that Junior may well become too large for its original case and will therefore require a new one – many readers may welcome the change!

Chapter 11 describes the cassette interface. This includes everything that is needed in the way of hardware and a brief description of the software (the software concerned is considered in greater detail in Book 4) required to transfer data (programs) to and from cassette tape. The cassette interface allows programs which would normally be stored in RAM, to be preserved in a more permanent manner, so that they can always be entered again, modified or further elaborated at a later date.

Additional extensions include an ASCII keyboard and a video interface or printer which are described in chapter 12. Programs can then be displayed on a TV screen (via the Elekterminal) or printed on paper. This gives greater scope for examining and modifying programs than the original monitor program could provide. These extensions are essential if the Junior Computer is to be operated in high level languages such as BASIC.

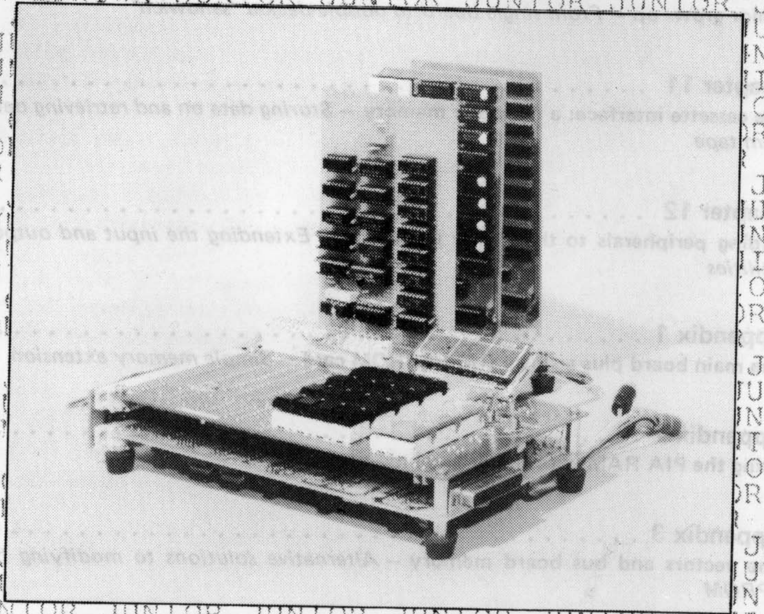
Each chapter provides clear and practical examples to help readers along the highways of programming.

And that's not all!

Book 4 is in actual fact the second half of Book 3 and contains:

- A new system program: the PM editor (PME)
- The PM software
- The PME software
- The TM software
- Extensive listings of all the system programs

Contents



10

Junior grows up

From single board to double-decker 'sandwich'

This opening chapter in Book 3 introduces a number of 'cards' (or printed circuit boards) which may be added to the basic Junior Computer for it to develop into maturity. One way to look at the fully-fledged system is to see it as a card game in which certain rules have to be kept if the system is to work correctly. Once all the cards have been 'laid on the table', the adult Junior Computer looks like this:

- **Additional 'brain-power' in the form of RAM and/or EPROM, up to a maximum of 64 kilo-bytes, is provided.**
- **It is equipped to pass data to and from magnetic tape (cassette interface).**
- **The computer is now able to communicate with the outside world in greater depth thanks to the addition of the 'Versatile Interface Adapter' (VIA) which has further input/output (I/O) facilities and allows more peripheral equipment, such as an ASCII keyboard and a video terminal or suitable printer to be incorporated.**

The interface board plays a key role in these new developments – in fact, it acts as the trump card. It will be described in detail during the course of this chapter, along with full constructional details. The other extension boards for the Junior Computer are based partly on existing cards which have already been published in Elektor magazine. One of these is the RAM/EPROM memory

card which regular readers will remember from the September 1980 issue. This has now been modified especially for the Junior Computer.

It should be noted at this stage in the proceedings (before you are up to your ears in solder and loose components, that is!) that readers are under no obligation to construct all the proposed extension boards — at least, not all at once. Memory cards have the advantage that they can be added one by one, over a period of time which the reader can spin out to suit his/her requirements (or his/her bank balance!). This subject will also come up for discussion later on.

Junior is growing . . . to adulthood . . . to maturity . . . to suit the growing needs of the enthusiastic programmer.

Growth is a natural process — it is part of nature. Junior Computer users share the same nature, in that, once they have become 'attached' to the machine, their interest and affection grow along with their knowledge. The desire for development and improvement becomes insatiable! As in all forms of growth, this involves a certain amount of time and, above all, patience.

In Book 1, the computer was compared to a human being, the hardware being regarded as its flesh and blood and the software as its brain. After a certain age, the physical growth of a human being reaches completion, whereas, ideally speaking, his/her intellectual development should continue until death. To put it in relative terms for the Junior Computer: the hardware necessary to prepare the machine for adult life must now be provided. What hardware is required? Once fully extended, the Junior Computer may be used with various peripheral devices (one or two cassette recorders, an ASCII keyboard, and either a video terminal or a printer — or both!). Before these additions can be made, however, the Junior Computer must be prepared 'mentally', that is, certain software must be developed first. With the aid of one or more of the system programs to be discussed later on, the computer is able to expand its vocabulary, its command of machine language. Eventually, it could even become bilingual, thinking in its 'mother tongue', machine language, but expressing its thoughts, either on paper or on a video screen, in a higher level language such as BASIC. What is important to note here is the fact that, similar to the human brain, the cultivation of computer software (in the form of additional system programs, other high level languages etc.) should theoretically carry on indefinitely, without the need for any further hardware extensions once the ones mentioned here have been incorporated into the basic system.

All a-board!

Buses, cards and bus boards

Although we said that growth is a natural process, in this particular case this is not strictly true. Obviously, the Junior Computer is not able to grow of its own accord — the user will have to lend it a helping hand. And a good thing too! After all, the world does not wish to be saddled with a digital Frankenstein! It is imperative that the user is able to gear any development to suit his/her personal requirements. It is therefore entirely up to the user whether or not the various extension possibilities suggested here are deemed necessary, and if so, he/she is free to decide on the number of stages and the order in which they are to be added. For this reason, it is best for everyone to consider carefully whether the extensions are needed and, if they are, which particular ones.

Readers should not let themselves be influenced by such considerations as: 'Everyone else does it . . .', 'Must keep up with the Joneses . . .' etc. To help you make up your mind, you are invited on a bus trip 'à la carte'.

A single or a double-decker system?

To bus or not to bus . . .

Computer systems designed for amateur use fall into two main categories: bus systems and single board systems. The former employs several cards (printed circuit boards) to accommodate all the necessary components. Usually, the cards are all the same size (the eurocard format is 100 mm x 160 mm, for instance) and are interconnected by means of a 'bus'. This is a printed circuit network in which equally positioned points (such as connector pins) are individually linked together. Buses are sometimes constructed on the basis of wire links and sometimes 'bus boards' are used. Since the bus is universally compatible within the system, the sky is the limit as far as the **theoretical** expansion possibilities are concerned. The final result is a computer with multiple facilities.

The one distinguishing feature of a single board system, on the other hand, is that all the components are mounted on the same card. The facilities available to the user determine the size and complexity of the card. As readers may well imagine, putting everything on a single board is a bit of a gamble. It means the user has to estimate his/her needs very carefully from the start and this often proves to be quite a handicap, as it is difficult to plan ahead when the possibilities are largely unknown.

Nevertheless, single board systems serve a variety of useful purposes. They are often used by apprentice programmers in order to gain 'hands on' experience and as part of relatively straightforward process control or monitor equipment (where the computer is switched on permanently). In its basic form the Junior Computer is in fact a single board system, being designed as an aid for beginners. Many enthusiasts are currently using it to sharpen their digital reflexes. So much is written and broadcast about 'chips' and 'microprocessors' nowadays that construction of 'your very own system' at home has become more than a passing phase: it is a once-in-a-lifetime milestone that many people feel they should reach before it gets too late.

The basic Junior Computer is already being used in many applications, ranging from analogue-to-digital conversion to process control in the manufacture of semiconductors. Therefore, the interface board is a purely optional addition and if desired, the expansion connector may be left untouched.

Growth: in which direction?

Nevertheless, the expansion connector is there and for several good reasons. It allows for:

More memory. Additional RAM is provided for user programs and for temporary storage of the variable output data pertaining to system programs including those recorded on magnetic tape. Additional EPROM is also made available to (permanently) store supplementary system programs. These could comprise several specific routines such as the TAPE MONITOR (TM) and PRINTER MONITOR (PM) routines which are discussed in chapters 11 and 12 respectively.

Additional I/O. The existing Peripheral Interface Adapter (PIA) is complemented by a colleague, the Versatile Interface Adapter (VIA), thus enabling a larger amount of data to be transferred to and from the outside world along a 'dual carriage-way'.

Cassette interface. This is a further hardware addition which allows ordinary magnetic tape to be used as **data archives** (background information) and as a **data library**. As their names suggest, permanent or temporary information (complete programs if necessary) is stored in the archives, whereas the library is filled with reference material, programs and subroutines which the user has developed, borrowed, copied or bought. These could well include complete operational programs, system programs (not necessarily resident in EPROM) or even a high level language interpreter such as BASIC. Since data transfer here is also bi-directional, a read/write memory (RAM) is an absolute must, even though the contents of an EPROM may be stored on tape. Naturally, the cassette interface hardware will have to have its own relevant software. This takes the form of the system program called TAPE MONITOR (TM) which was mentioned in passing earlier and which will be dealt with in greater detail in chapter 11.

RS 232 interface. Additional hardware is required to set up communication channels along which data can be transferred between the computer and virtually any complex peripheral device. However, the data 'traffic flow' must abide by certain regulations, depending on the peripheral device(s) used. To start with, Junior Computer owners are advised to replace (or rather supplement) the standard hexadecimal keyboard with a full-scale ASCII version. The output which formerly appeared on six seven-segment displays can now best be expressed either on paper (using a suitable printer) or on a television screen (video terminal). The Elekterminal and ASCII keyboard published in the November and December 1978 editions of Elektor (E43 and E44) are highly recommended for this.

Once a computer is able to provide various modes of expression, its 'vocabulary' can be extended to include a number of new system programs, such as a text editor, an assembler, a disassembler (from the hex dump to the complete listing), etc. One system program mentioned earlier has already been developed and is ready for use. This is the PRINTER MONITOR (PM) routine and will receive full attention in chapter 12.

That covers all the basic details of the hardware required to convert the Junior Computer to run on a high level language such as BASIC. Once the expansion connector is employed, the Junior Computer is no longer a single board system. This does not mean that its possibilities as a bus system are infinite. On the contrary, the computer is limited by its own hardware. In fact, the only hardware addition that is envisaged after Book 3 is a 16k dynamic RAM card. Strictly speaking, this is not a hardware expansion at all, but an economic alternative to the RAM/EPROM cards described in the September 1980 issue of Elektor (E65) and later on in this chapter.

In any case, the extended version of the Junior Computer bears no real resemblance to the average bus system. All the extensions mentioned above are included on the actual interface board. This has been made the same size as the main board of the Junior Computer, so that the two can be 'sandwiched' together. If required, an existing SC/MP bus board (EPS number: 80024, see the January 1980 issue of Elektor) may be linked to the interface card to house several extra memory boards.

In spite of the bus board addition, therefore, the Junior Computer is not a true bus system, but rather a 'double-decker' sandwich! However, never mind the exact category the computer falls into, let us examine the system itself in closer detail.

The interface board . . .

. . . the trump card!

Apart from the optional peripheral devices, the majority of the extension facilities are housed on the interface board. Readers with a healthy appetite for bytes are, of course, welcome to add further bus boards and memory cards (up to a maximum of 64 k).

The word 'interface' means 'link'. It can be thought of as the physical transition from computer childhood to adulthood. Here the interface board provides the Junior Computer with a vital link, like an umbilical cord, with the outside world. This life-line consists of various communication channels: additional I/O, a cassette interface, an RS 232 interface and an internal connection to address and data buses to provide a buffered 'highway'.

All the electronics involved may be found in figures 1 and 2. As can be seen, it is more elaborate than the main board even though it is equally compact in size. Each component will now be discussed separately.

The buffers: softening electronic blows

All the connections to the INPUT CONNECTOR are shown on the left-hand side of figure 1. This connector must be linked to the EXPANSION CONNECTOR belonging to the basic Junior Computer. As it happens, the two boards are linked by several such 'umbilical cords'. One of these is made up of five connections to the PORT CONNECTOR of the main

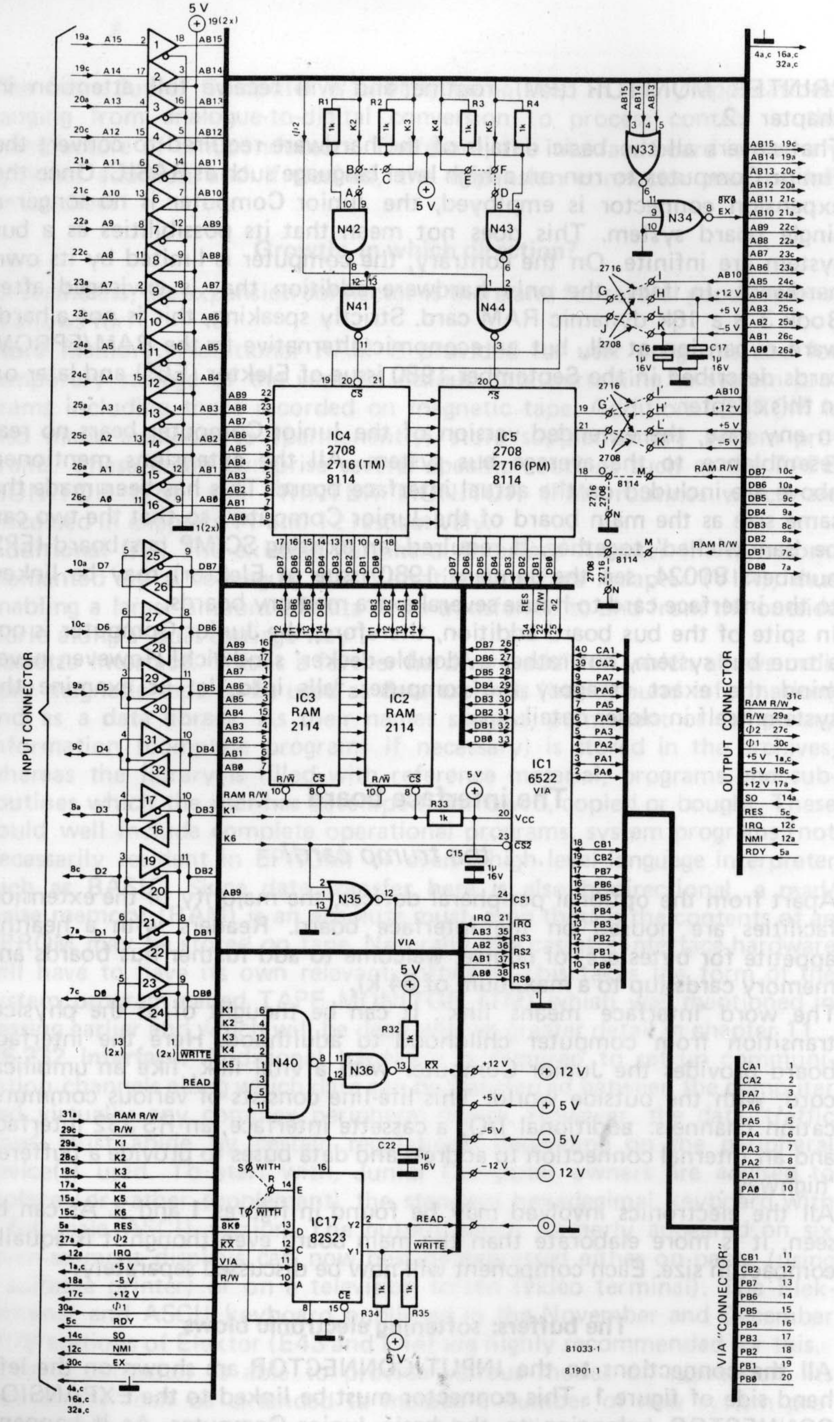
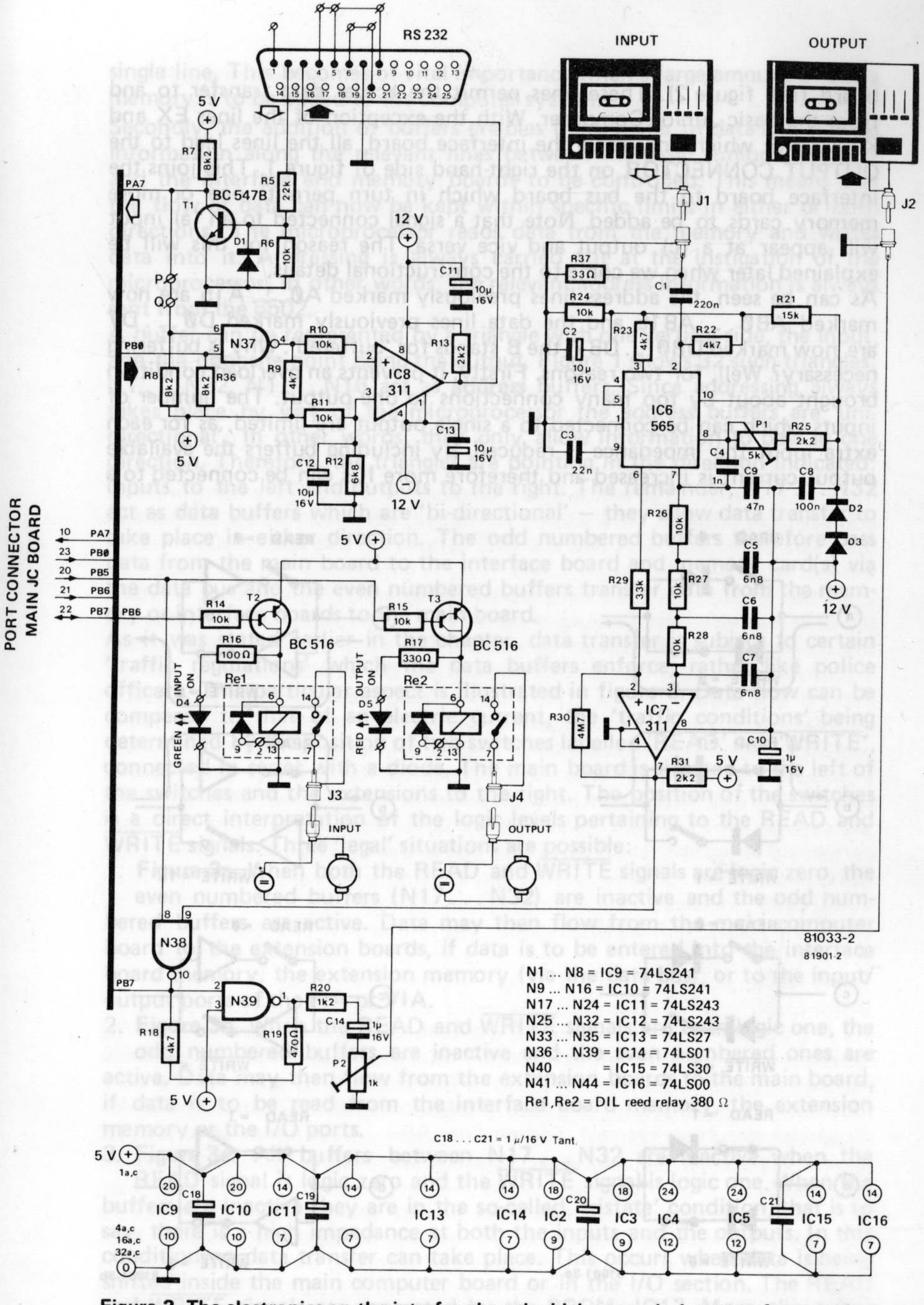


Figure 1. The circuit diagram of the main section of the interface board. It provides more memory, additional I/O, buffering including data buffer control and complete address decoding.



RS 232
INPUT
OUTPUT
J2
PORT CONNECTOR MAIN JC BOARD
BC 547B
BC 516
IC8 311
IC6 565
IC7 311
Re1
Re2
J3
J4
INPUT
OUTPUT
N37
N38
N39
GREEN INPUT ON
RED OUTPUT ON
81033-2
N1 ... N8 = IC9 = 74LS241
N9 ... N16 = IC10 = 74LS241
N17 ... N24 = IC11 = 74LS243
N25 ... N32 = IC12 = 74LS243
N33 ... N35 = IC13 = 74LS27
N36 ... N39 = IC14 = 74LS01
N40 = IC15 = 74LS30
N41 ... N44 = IC16 = 74LS00
Re1, Re2 = DIL reed relay 380 Ω
C18 ... C21 = 1 µ/16 V Tant
IC9
IC10
IC11
IC13
IC14
IC2
IC3
IC4
IC5
IC15
IC16

Figure 2. The electronics on the interface board which controls data transfer to and from magnetic tape and to and from peripheral devices. This section contains the cassette interface and the RS 232 interface.

board (see figure 2). These lines permit the actual data transfer to and from the basic Junior Computer. With the exception of the lines EX and K1 . . . K6, which only serve the interface board, all the lines lead to the OUTPUT CONNECTOR on the right-hand side of figure 1. This joins the interface board to the bus board which in turn permits one or more memory cards to be added. Note that a signal connected to an (a) input will appear at a (c) output and vice versa. The reason for this will be explained later when we come to the constructional details.

As can be seen, the address lines previously marked AØ . . . A15 are now marked ABØ . . . AB15 and the data lines previously marked DØ . . . D7 are now marked DBØ . . . DB7; the B stands for 'buffered'. Why is buffering necessary? Well, for two reasons. Firstly, it prevents an overload condition brought about by too many connections to one output. The number of inputs which can be connected to a single output are limited, as for each extra input the impedance is reduced. By including buffers the available output current is increased and therefore more ICs can be connected to a single line. This becomes of vital importance when a large amount of extra memory is to be added to the computer system.

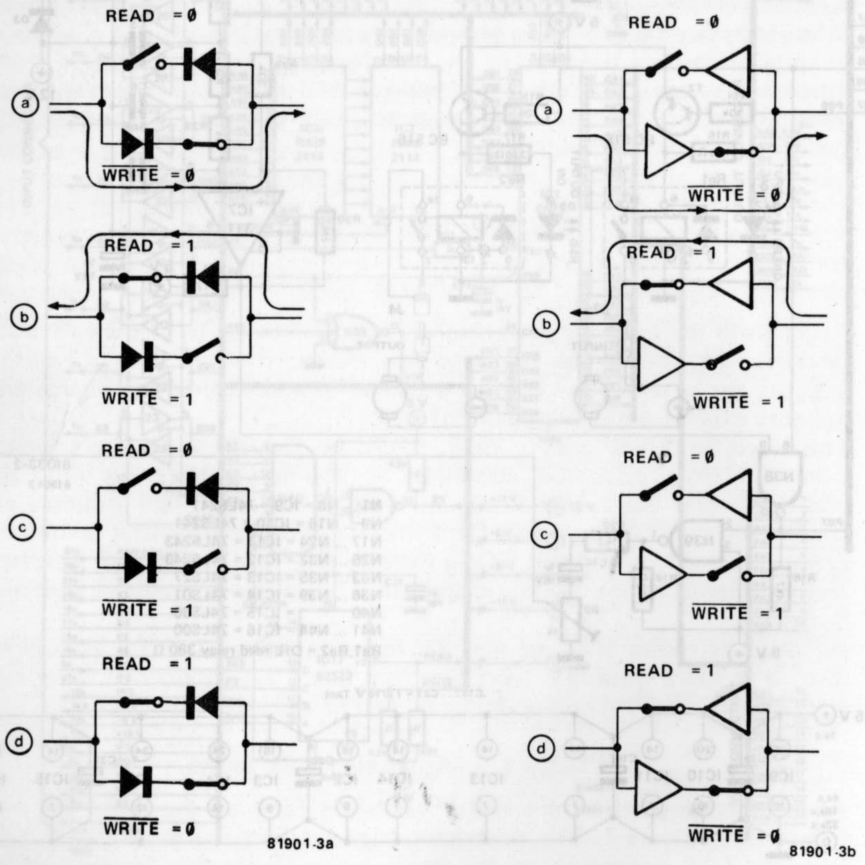


Figure 3. The principle of operation of the data buffers is illustrated here by means of diodes and switches.

Secondly, the addition of buffers enables the transfer of data and address information along the relevant lines between the basic Junior Computer and the interface and memory boards to be controlled. This means that the flow of data can now be kept within specific limits in either or both directions. The microprocessor reads data from the memory and writes data into it. Addressing is always carried out at the instigation of the microprocessor, in other words, the relevant address information is always sent **from** the 6502.

A buffer can be represented by a triangle with one vertical side, the input, and the opposite point being the output. Figure 1 shows 32 such triangles: N1 . . . N32. N1 . . . N16 act as **address** buffers. Since addressing always takes place by way of the microprocessor the address buffers are 'unidirectional'. In other words, they only allow information to pass in one direction. Therefore, the triangles are pointing in the direction indicated: inputs to the left and outputs to the right. The remainder, N17 . . . N32 act as **data** buffers which are 'bi-directional' — they allow data transfer to take place in either direction. The odd numbered buffers therefore pass data from the main board to the interface board and memory card(s) via the data bus and the even numbered buffers transfer data from the memory or interface boards to the main board.

As it was stated earlier in the chapter, data transfer is subject to certain 'traffic regulations' which the data buffers enforce, rather like police officers. This particular aspect is illustrated in figure 3. Data flow can be compared to that of an electric current, the 'traffic conditions' being determined by the position of two switches labelled 'READ' and '$\overline{\text{WRITE}}$', connected in series with a diode. The main board is situated to the left of the switches and the extensions to the right. The position of the switches is a direct interpretation of the logic levels pertaining to the READ and $\overline{\text{WRITE}}$ signals. Three 'legal' situations are possible:

1. **Figure 3a.** When both the READ and $\overline{\text{WRITE}}$ signals are logic zero, the even numbered buffers (N17 . . . N32) are inactive and the odd numbered buffers are active. Data may then flow from the main computer board to the extension boards, if data is to be entered into the interface board memory, the extension memory (via the data bus), or to the input/output ports of the PIA or VIA.
2. **Figure 3b.** When the READ and $\overline{\text{WRITE}}$ signals are both logic one, the odd numbered buffers are inactive and the even numbered ones are active. Data may then flow from the extension boards to the main board, if data is to be read from the interface board memory, the extension memory or the I/O ports.
3. **Figure 3c.** All buffers between N17 . . . N32 are inactive when the READ signal is logic zero and the $\overline{\text{WRITE}}$ signal is logic one. When the buffers are inactive they are in the so-called 'tristate' condition, that is to say, there is a high impedance at both the inputs and the outputs. In this condition no data transfer can take place. This occurs when data is being shifted inside the main computer board or in the I/O section. The READ and $\overline{\text{WRITE}}$ signals are generated by the PROM, IC17. More about this device later on.

N.B. Figure 3d shows a fourth, theoretical, possibility where the READ signal is logic one and the $\overline{\mathrm{WRITE}}$ signal is logic zero, all the buffers being active at the same time. Needless to say, this situation should never occur, as data may not be read and written simultaneously. This is again taken care of by the PROM, IC17.
(Note that the points of the diodes (cathodes) in figure 3 correspond to those of the triangles in figure 1).

The Versatile Interface Adapter (VIA)

Additional I/O

The Versatile Interface Adapter type 6522, IC1, merits individual attention. In fact a whole chapter is devoted to this particular device in Book 4. The VIA is remarkable in that it offers more facilities than the standard I/O device, the 6532 PIA. As shown in figure 1, the VIA 'CONNECTOR' incorporates the relevant connections to the outside world. (It is put in inverted commas, as it is not a real connector). The 6522 is controlled by address lines ABØ . . . AB3 and by various signals presented via the control bus. The eight data lines and the IRQ output (to the left of IC1) will be familiar from Book 1.
Like the 6532, the 6522 features two chip select inputs. One of them ($\overline{\mathrm{CS2}}$) is connected to the output signal K6 of the address decoder, IC6, on the main board of the Junior Computer. The other (CS1) is linked to the output of gate N35 which is controlled by the select signal K6 and the address line AB9. For the VIA to be enabled, the chip select inputs CS1 and $\overline{\mathrm{CS2}}$ will have to be logic one and logic zero respectively. Gate N35 is a NOR gate, which means that its output will only be high when all the inputs are low. Effectively, this means that both select line K6 and address line A9 have to be logic zero before the VIA will be enabled (in the case of the PIA, A9 has to be logic one). Since address line A8 is not connected to either the 6522 or the 6532, the VIA will be situated in the following address range:
18ØØ/19ØØ . . . 18FF/19FF
(AB8 = x; AB9 = Ø; K6 = Ø)
and the PIA:
1AØØ/1BØØ . . . 1AFF/1BFF
(A8 = x; A9 = 1; K6 = Ø)
As the address lines A8 and AB8 are both the same (x = either Ø or 1), multiple addressing is eliminated and a total of 256 addresses are available for both the PIA and the VIA. In the former case, 19 different memory locations are provided for the PIA in addition to the 128 bytes of RAM. As can be seen from figure 1, address lines AB4 . . . AB7 are not connected to the VIA. Thus there are only 16 different memory locations available for this IC.
Why is an additional input/output nucleus necessary? Well, the PIA already has plenty of internal 'housekeeping' to cope with, such as controlling the six seven-segment displays and scanning the hexadecimal keyboard. Once the RS 232 and cassette interfaces are added, the PIA will be assigned several other tasks as well. As a result, certain restrictions have to be imposed on programs using the PIA in combination with the monitor routines. If, for instance, one of the programs from Book 2, chapter 6, is to be examined in the step mode, all sorts of things could go wrong (let alone the timing!). Stepping through a program involves a jump to the SAVE routine of the monitor program after each instruction. Readers who have forgotten this aspect had better refer back to chapter 7 in Book 2! The slightest change to the normal state of the input/output ports of the PIA will affect the operation of the monitor program considerably and it may even stop working altogether.
This is just one of the many dangers involved, but they can all be avoided by simply reserving the PIA for the internal affairs of the computer and by keeping the VIA for use in personal programs and other applications. This does not mean, of course, that monitor subroutines cannot be incorporated into user programs.

Address decoding

Certain groups of addresses in the Junior Computer (memory locations) share one common feature: a specific select signal. Memory on both the main and the interface boards, as will be seen later on, is selected and addressed by the signals KØ . . . K7. These signals are produced by the address decoder, IC6, on the main board. Additional memory which is connected via the bus board is also selected by similar signals which are generated on the memory board(s) itself (themselves). Further details of this will be provided in the description of the RAM/EPROM board towards the end of this chapter.
At this stage it might be a good idea to recap on the exact function of IC6, referring back to chapter 1, Book 1. The operation of the address decoder is illustrated in terms of hardware in figure 4, where it is shown as a switch, and in terms of software in table 1 in the form of a truth table. Whenever a certain select signal in the range KØ . . . K7 is chosen, the selected K output will go low (logic zero) and the other seven will all be pulled up to the positive supply voltage via the resistors, and will therefore be logic one. As you will probably remember, chip select inputs are usually labelled $\overline{\mathrm{CS}}$, where the 'bar' indicates that the device is enabled with an active low (logic zero) signal.
As a matter of fact, the switch in figure 4 does not feature eight, but sixteen different positions. The other eight contacts are not connected. Effectively, the position of the switch is determined by the three address lines A13 . . . A15 and point 'D' of IC6, which is either connected to ground (= logic zero) or connected to point 'EX' on the expansion connector (the wire link on the main board is moved to point EX).
Whenever point 'D' is grounded one of the contacts Ø . . . 7 must have been selected. If on the other hand, point 'D' is connected to point 'EX', the logic level present on the latter will determine whether a contact between Ø . . . 7 (EX = logic Ø) or a disconnected contact between 8 . . . 15 (EX = logic 1) is selected. Let us now take a look at the right-hand column in table 1.
Clearly, whenever point D is logic zero, something is being decoded either on the main board or on the interface board. Whenever point D is logic one, however, this means that memory is being accessed on one of the

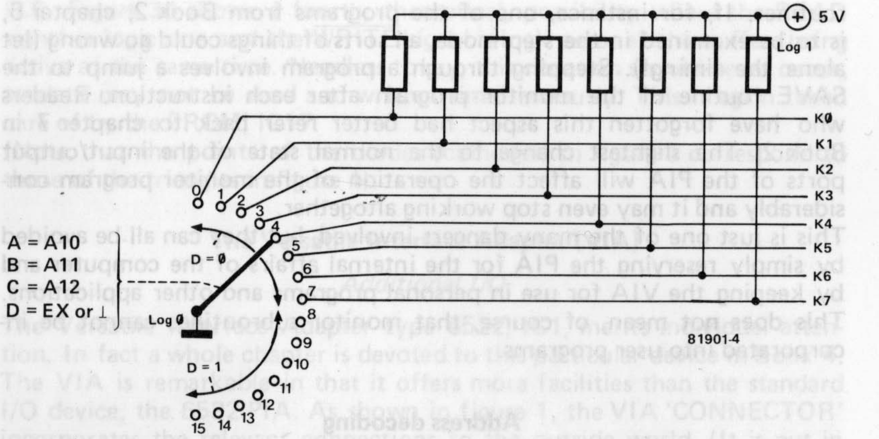


Figure 4. A general idea of what goes on inside the address decoder IC6 on the main board. The eight contacts enable memory situated on the main and interface boards.

Table 1. The truth table for the address decoder (IC6) on the main board.

D = ⊥ (φ) or D = EX = $\overline{8K\emptyset}$	C = A12	B = A11	A = A10	K7	K6	K5	K4	K3	K2	K1	KØ	m = main board i = interface board b = bus board
Ø	Ø	Ø	Ø	1	1	1	1	1	1	1	Ø	m
Ø	Ø	Ø	1	1	1	1	1	1	1	Ø	1	i
Ø	Ø	1	Ø	1	1	1	1	1	Ø	1	1	i
Ø	Ø	1	1	1	1	1	1	Ø	1	1	1	i
Ø	1	Ø	Ø	1	1	1	Ø	1	1	1	1	i
Ø	1	Ø	1	1	1	Ø	1	1	1	1	1	i
Ø	1	1	Ø	1	Ø	1	1	1	1	1	1	m + i
Ø	1	1	1	Ø	1	1	1	1	1	1	1	m
1	Ø	Ø	Ø	1	1	1	1	1	1	1	1	b
1	Ø	Ø	1	1	1	1	1	1	1	1	1	b
1	Ø	1	Ø	1	1	1	1	1	1	1	1	b
1	Ø	1	1	1	1	1	1	1	1	1	1	b
1	1	Ø	Ø	1	1	1	1	1	1	1	1	b
1	1	Ø	1	1	1	1	1	1	1	1	1	b
1	1	1	Ø	1	1	1	1	1	1	1	1	b
1	1	1	1	1	1	1	1	1	1	1	1	b

additional cards connected to the bus board. Obviously, for this to be the case, point D on the main board will have to be connected to point EX, for it to become logic one. But, if no extra memory cards are to be employed, the wire link from point D may be grounded. For then the select (K) outputs of IC6 can be used to access any available interface board memory up to a maximum of 5k.

Interface board memory

Additional RAM and/or EPROM

The memory extension on the interface board covers several ICs and may be supplemented with bus board memory. We will leave the latter for the time being and concentrate for the moment on the amount of memory available on the interface board. The memory areas available on the main board are selected by lines KØ and K7 and are absolutely full to the brim. Select line K6 serves the input/output section and lines K1 . . . K5 are reserved to access the total, additional memory areas included on the interface board. Each select signal can access up to 1k of memory, which means that up to 5k is available. This allows the use of up the 8k total which is decoded by the basic Junior Computer (see table 1), including the 1k for the PIA and VIA.

In the first place, additional RAM is available. The RAM situated on the main board is located on pages ØØ . . . Ø3.Even without using other extension facilities, the operator can now make use of 1k of RAM on the interface board (IC1 and IC2). This particular section of RAM is selected by means of line K1. Therefore the address range is Ø4ØØ . . . Ø7FF. In other words, four further pages (Ø4 . . . Ø7) are available which link up nicely with the RAM on the main board (pages ØØ . . . Ø3) so that lengthy user programs can now be entered 'in one piece'.

Both IC4 and IC5 may be 1k RAMs (type 8114), 1k EPROMs (type 2708) or 2k EPROMs (type 2716). One or two select signals, K2 . . . K5, enable each IC with the aid of the clock signal Ø2 and gates N41 . . . N44. The clock signal is necessary to time the read cycle and (if appropriate) the write cycle accurately.

The address ranges for the various select lines are as follows:

K2 = logic Ø – addresses Ø8ØØ . . . ØBFF
K3 = logic Ø – addresses ØCØØ . . . ØFFF
K4 = logic Ø – addresses 1ØØØ . . . 13FF
K5 = logic Ø – addresses 14ØØ . . . 17FF

Various selection possibilities are provided depending on the particular type of device used for IC4 and IC5. If 8114s (1k of RAM) or 2708s (1k of EPROM) are used, only one select input is required per IC. If 2716s (2k of EPROM) are used two select signals have to be combined: K2 and K3 for IC4 (address range Ø8ØØ . . . ØFFF) and K4 and K5 for IC5 (address range 1ØØØ . . . 17FF). Solder points A . . . F are provided to select the required address range and are interconnected to suit the type of memory used. The same is true for points G . . . O and G' . . . O', which are wired according to the value of the supply voltage requirements and also according to whether or not an additional address line (AB 1Ø) or the RAM-R/W signal is required.

Full or partial address decoding?

Sixteen or thirteen address lines?

In the basic version of the Junior Computer only 8k out of a possible total 64k of memory is actually decoded. Address lines A13 . . . A15 are not used, leaving thirteen address lines to access 2^{13} = 8192 (8k) different memory locations. This is, of course, none other than the 8k of memory which can be enabled with the eight (K) outputs of the address decoder. Earlier we stated that the full 8k can be utilised, once the interface board has been connected up to the main computer board. Thus, provided we are prepared to get by without utilising any extra memory cards, there will be no need to use address lines A13 . . . A15.

However, supposing that we do wish to use some extra memory after all, the 8k so far available is fully occupied and so we will have to draw upon the remaining 56k (64 – 8). The remaining three address lines are now required and the following must take place:

1. Special provisions will have to be made to ensure that only the extra memory is accessed and that the original, basic 8k remains untouched (see table 1). This is necessary to avoid multiple addressing which can lead to disastrous results. The solution is quite simple: connect point D to point EX (move the wire link from ground to EX). This will ensure that none of the select outputs from IC6 will be low whenever point EX is logic one.
2. Supplementary address decoding is required on the RAM/EPROM card(s). Again, this will be considered in greater detail towards the end of this chapter. In the top right-hand corner of figure 1, point EX is shown connected to the output of an inverter, N34. This inverter is controlled via the NOR gate N33 which is, in turn, connected to address lines AB13 . . . AB15. As soon as one of these three address lines becomes logic one, point EX and therefore point D of IC6 will also become logic one. When all three address lines AB13 . . . AB15 are logic zero, point EX and point D will also be logic zero. The end result is the following address system:

The total main and interface board memory and the I/O (PIA and VIA) can be accessed between address locations: ØØØØ . . . 1FFF (pages ØØ . . . 1F) = 32 pages = 8k.

Depending on the number of extra memory cards required (up to 56k), one or more memory areas can be selected in the address range:

2ØØØ . . . FFFF = 256 – 32 = 224 pages of ¼k each.

Partial address decoding (without address lines AB13 . . . AB15), as you know, causes **multiple addressing**. An address is always expressed as four hexadecimal figures (= nibbles). The 'superfluous' address lines A13 . . . A15 correspond to three-quarters of the most significant nibble, in other words, to the left-hand nibble of the page to which the address belongs. Since any of these address lines can go high without the operator's knowledge, each individual address will appear eight times.

Consequently, addresses belonging to page Øx where x = Ø . . . F, will also appear in pages 2x, 4x, 6x, 8x, Ax, Cx and Ex. Conversely, addresses belonging to page 1x where x = Ø . . . F, are also accessed on pages 3x, 5x, 7x, 9x, Bx, Dx and Fx.

Thus, as soon as the memory is extended beyond the 8k, the two provisions mentioned above have to be made, enabling any address location within the range 2ØØØ . . . FFFF to be accessed only once.

Data traffic control

How to manipulate the data buffers

The EX output signal is also passed, under the nomenclature $\overline{8KØ}$, to one of the address inputs of the PROM, IC17. This device contains 32 bytes in all, but only two bits in each byte, Y1 (= $\overline{\text{WRITE}}$) and Y2 (= READ), are used. These two bits control the direction of data flow through the buffers N17 . . . N32. The 32 bytes can be accessed via the five address line inputs A . . . E, which will be described later on.

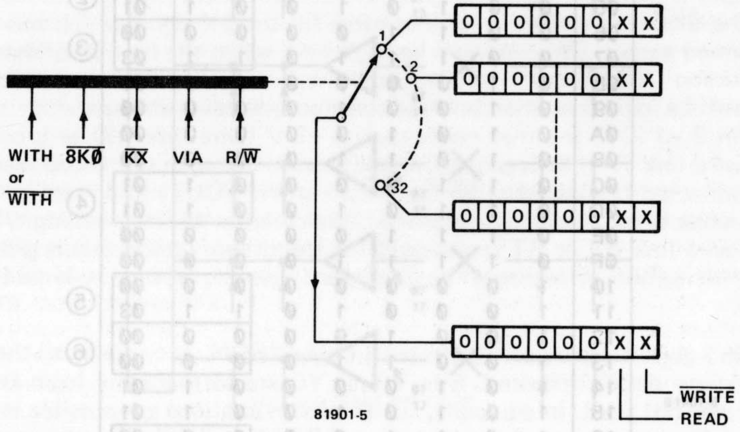


Figure 5. The PROM, IC17, represented as a 'byte switch'. Five address lines select one of the possible 32 positions. However, not all 32 positions are actually used.

The operation principle of the PROM is illustrated in figure 5. A 32 position switch is used to select one of the 32 possible data bytes. The required position being determined by the logic levels present on the five address lines. The result is a particular 8 bit (= 1 byte) 'word' at the eight data outputs. Obviously, the 32 bytes have to be programmed into the PROM beforehand. The required patterns of 'ones' and 'noughts' are listed in table 2. Provided the five address lines all feature a specific logic level, a certain bit pattern will be constantly available at the output of the PROM. In other words, the PROM is read continuously, without the need for a chip select ($\overline{\text{CS}}$) or read/$\overline{\text{write}}$ (R/$\overline{W}$) control signal.

Seeing as the PROM is so complicated to access, why have one at all? Why not simply make do with the R/$\overline{W}$ signal in the manner provided on the main board of the Junior Computer? To find out, let us see what happens when data is read from the RAM, EPROM or I/O situated on the main board. If the R/$\overline{W}$ signal is connected directly to the READ and $\overline{\text{WRITE}}$

Table 2. The contents of the PROM, IC17. The 'program' is shown as hexadecimal addresses and data.

PROM-address (hex)	E = WITH of $\overline{\text{WITH}}$	D = $\overline{\text{8KØ}}$	C = $\overline{\text{KX}}$	B = VIA	A = R/W	Y3 . . . Y8	Y2 = READ	Y1 = $\overline{\text{WRITE}}$	PROM-data (hex)	
ØØ	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	①
Ø1	Ø	Ø	Ø	Ø	1	Ø	1	1	Ø3	
Ø2	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	ØØ	
Ø3	Ø	Ø	Ø	1	1	Ø	Ø	Ø	ØØ	
Ø4	Ø	Ø	1	Ø	Ø	Ø	Ø	1	Ø1	②
Ø5	Ø	Ø	1	Ø	1	Ø	Ø	1	Ø1	
Ø6	Ø	Ø	1	1	Ø	Ø	Ø	Ø	ØØ	③
Ø7	Ø	Ø	1	1	1	Ø	1	1	Ø3	
Ø8	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	
Ø9	Ø	1	Ø	Ø	1	Ø	Ø	Ø	ØØ	
ØA	Ø	1	Ø	1	Ø	Ø	Ø	Ø	ØØ	
ØB	Ø	1	Ø	1	1	Ø	Ø	Ø	ØØ	
ØC	Ø	1	1	Ø	Ø	Ø	Ø	1	Ø1	④
ØD	Ø	1	1	Ø	1	Ø	Ø	1	Ø1	
ØE	Ø	1	1	1	Ø	Ø	Ø	Ø	ØØ	
ØF	Ø	1	1	1	1	Ø	Ø	Ø	ØØ	
1Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	⑤
11	1	Ø	Ø	Ø	1	Ø	1	1	Ø3	
12	1	Ø	Ø	1	Ø	Ø	Ø	Ø	ØØ	⑥
13	1	Ø	Ø	1	1	Ø	Ø	Ø	ØØ	
14	1	Ø	1	Ø	Ø	Ø	Ø	1	Ø1	
15	1	Ø	1	Ø	1	Ø	Ø	1	Ø1	
16	1	Ø	1	1	Ø	Ø	Ø	Ø	ØØ	⑦
17	1	Ø	1	1	1	Ø	1	1	Ø3	
18	1	1	Ø	Ø	Ø	Ø	Ø	Ø	ØØ	
19	1	1	Ø	Ø	1	Ø	Ø	Ø	ØØ	
1A	1	1	Ø	1	Ø	Ø	Ø	Ø	ØØ	
1B	1	1	Ø	1	1	Ø	Ø	Ø	ØØ	
1C	1	1	1	Ø	Ø	Ø	Ø	Ø	ØØ	⑧
1D	1	1	1	Ø	1	Ø	1	1	Ø3	
1E	1	1	1	1	Ø	Ø	Ø	Ø	ØØ	
1F	1	1	1	1	1	Ø	Ø	Ø	ØØ	

connections on the interface board, the odd numbered data buffers, N17 . . . N32, will be enabled each time a write operation occurs and the even numbered buffers will be enabled at every read operation. This means that they will allow data to flow to and from the data bus even though the interface board itself is not necessarily being addressed. Thus, as the inputs will be in a random logic state when the outputs are activated, the information appearing on the data lines is bound to be incorrect. The solution, therefore, is to control the data buffers with the aid of the address system of each circuit, so that data transfer through the buffers only takes place

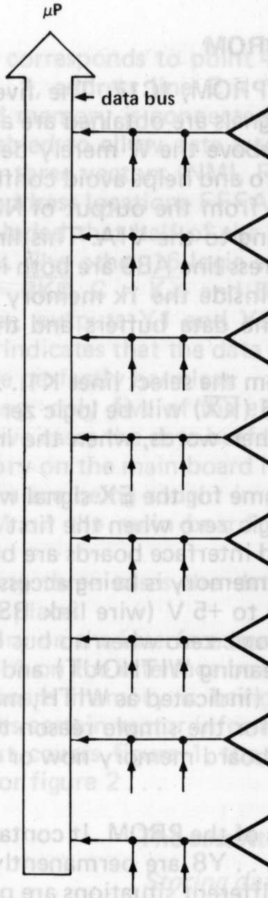


Figure 6. The undesirable situation which crops up if the processor reads an addressable memory range on the main board and the eight data buffers are enabled. This leads to 'multiple addressing'. This does not matter provided the eight data bits are at the same logic level as the corresponding buffer outputs, which is very unlikely! The only solution is to disable the data buffers in such instances.

at the correct moments.

The PROM is included for another reason as well. The NMI, RES and IRQ vectors can now (once the address decoding is complete) be stored at their true address locations FFFA . . . FFFF, instead of 1FFA . . . 1FFF (as in the EPROM of the basic version of the Junior Computer). This will only work, of course, if page FF is actually stored in EPROM. Since page FF is now included on a RAM/EPROM card, which is connected to the main computer board by means of a bus board, we will refer to it as bus board memory from now on.

N.B. As soon as the bus board is used to connect one or more memory extension cards, page FF must be included in EPROM with the correct vectors, so that multiple addressing (where FF = 1F etc.) can no longer occur.

The PROM

Now to get back to addressing the PROM, IC17. The five address lines from which the READ and $\overline{\text{WRITE}}$ signals are obtained are as follows:

A. The R/W or R/$\overline{\text{W}}$ signal (the bar above the W merely denotes that the write signal is active when logic zero and helps avoid confusion).

B. The 'VIA' signal. This is obtained from the output of N35 and is none other than the CS1 signal pertaining to the VIA. This line is logic one when the select line K6 and the address line AB9 are both logic zero, that is to say, when the VIA is selected. Inside the 1k memory zone decoded by K6, the PIA is situated before the data buffers and the VIA behind them.

C. The $\overline{\text{KX}}$ signal. This is derived from the select lines K1 . . . K5 via gates N40 and N36. The output of N36 ($\overline{\text{KX}}$) will be logic zero when any of the select lines are logic zero, in other words, when the interface board memory is being accessed.

D. The $\overline{\text{8KØ}}$ signal. This is another name for the EX signal which was mentioned earlier. This line will be logic zero when the first 8k of memory (including I/O) on both the main and interface boards are being addressed and will be logic one when bus board memory is being accessed.

E. Pin E of IC17 is either connected to +5 V (wire link RS) or grounded (wire link RT). This pin must be logic zero when no bus board memory is connected (indicated as $\overline{\text{WITH}}$, meaning WITHOUT) and must be logic one when extra memory is employed (indicated as WITH, meaning WITH). The wire link RS or RT is included for the simple reason that it gives the user the option of adding extra bus board memory now or at some future date.

Table 2 gives a survey of the contents of the PROM. It contains all the programming details required. Bits Y3 . . Y8 are permanently logic zero as they are not used. In principle, 32 different situations are possible leading to a selection of one of the three 'legal' READ and $\overline{\text{WRITE}}$ combinations. In practice, however, only eight situations remain if the separation into read and write is disregarded:

1. Writing to or reading from memory on the interface card (IC2 . . . IC5). The data buffers are enabled to provide a read or write operation in PROM addresses ØØ and Ø1.
2. Reading from EPROM and either reading from or writing to RAM or the PIA on the main board. The data buffers are all disabled (PROM addresses Ø4 and Ø5).
3. Reading from or writing to the VIA. Since this involves the interface board, the data buffers will be enabled to provide either process (PROM addresses Ø6 and Ø7).
4. Writing to or reading from bus board memory (PROM addresses ØC and ØD). Since in this situation address line E is logic zero ($\overline{\text{WITH}}$ = WITHOUT), no bus board memory is connected and so the data buffers must all be *disabled*. The NMI, $\overline{\text{RES}}$ and IRQ vectors are automatically defined by the EPROM on the main board of the Junior Computer.
5. See point 1 (PROM addresses 1Ø and 11).
6. See point 2 (PROM addresses 14 and 15).
7. See point 3 (PROM addresses 16 and 17).
8. This partly corresponds to point 4 (PROM addresses 1C and 1D). As in points 5 . . 7, address line E is now logic one, (WITH), which means that bus board memory is connected. Now, however, the data buffers will have to be enabled to allow data transfer to take place in one direction or the other. The three vectors (NMI, RES and IRQ) must now be contained in EPROM at address locations FFFA . . . FFFF.

It can be concluded that half of the 32 bytes inside the PROM, IC17, are truly necessary. The other 16 logic states are irrelevant, since such combinations as D = $\overline{\text{8KØ}}$, C = $\overline{\text{KX}}$ and B = VIA just do not arise. As table 2 shows, however, outputs Y1 and Y2 are always logic zero in the other 16 cases. This indicates that the data buffers are prepared for a write operation and so are perfectly harmless.

Strictly speaking, only five of the eight situations mentioned ever occur. In two of the five cases the data buffers are disabled:

a. When memory on the main board is being accessed;

b. When vectors are being sought in page FF, whilst referring to the original EPROM on the main board since no bus board memory is connected.

In the remaining three cases the data buffers are enabled to allow data transfer to take place:

c. When memory on the interface board is being accessed;

d. When the VIA on the interface board is being accessed;

e. When bus board memory is being accessed, because, for instance, the 6502 requires certain vector information.

That just about covers figure 1, the address decoding and buffer control systems, now for figure 2 . . .

The cassette interface

Storing data on tape

The majority of components shown in figure 2 refer to the cassette interface. This includes everything that is required in the way of hardware (the software involved is dealt with in chapter 11) to transfer data to and from a normal cassette recorder. When data is being read the tape recorder will be in the playback mode and when data is being written the cassette is in the record mode.

Data is transferred to and from the microprocessor by way of the PIA port line PB7 which is available on the port connector of the main board. When data is being written during the DUMP/DUMPT subroutine of the TAPE MONITOR system program, port line PB7 functions as an output, as do port lines PB5 and PB6. These latter two lines will then be logic zero and logic one respectively. As a result, the input of N38 (pin number 8) will also be low and so its output will be high (N38 has an open collector output which presents a high impedance whenever it is logic one). Since port line PB5 is logic zero, the PNP Darlington transistor T3 is turned on via resistor R15 and so the red LED, D5 (OUTPUT ON) lights and relay Re2 is activated. The relay contacts are linked to the output socket J4. If a corresponding plug is wired in series with the motor of the tape recorder, the latter can be switched on by means of software, provided of course the machine is ready to record.

Since port line PB6 is logic one during a write operation, transistor T2 will not conduct, the green LED, D4, will not light and relay Re1 will not be activated. The output of gate N39 is fed to potentiometer P2 via resistor R20 and capacitor C14 and is used to preset the maximum record level. Connector J2 is the actual data output socket.

When data is being read from tape during the subroutine RDTAPE contained in the TAPE MONITOR system program, port line PB7 functions as an input and lines PB5 and PB6 will once again be outputs. This time port line PB5 will be logic one and port line PB6 will be logic zero. As a result, gate N38 can now pass on the data output signal from IC7 to the port line PB7 after inverting it. Gate N39 remains enabled, so that the data signals are also passed to the output socket J2, but without doing any harm. Now that line PB5 is logic one, the red LED, D5, will not light. Port line PB6 will now be logic zero causing transistor T2 to conduct, the green LED, D4 (INPUT ON), to light and relay Re1 to be activated. The contacts of this relay are linked to socket J3 which can be used to control the playback machine.

There is no absolute need to utilise two tape recorders for storing and retrieving data. If only one is used for both purposes, remote control will no longer be necessary and the relays will then seem superfluous. However, it is best to mount them both on the board 'just in case'!

In addition, transistors T2 and T3 and the two LEDs provide useful information about what is actually going on. The red LED lights when data is being written to the cassette recorder and the green LED lights when data is being retrieved from tape.

The PLL

Digital to audio conversion (and vice versa)

The section of circuit in figure 2 between the data input socket, J1, and the input of N38 (pin number 9) is next on the list for discussion. The circuitry around IC6 and IC7 will undoubtedly look familiar to KIM operators. It is in fact similar to part of the KIM hardware, albeit improved, because we see no reason for re-inventing the wheel!

Before describing the circuit operation in detail, it is useful to know that the data is recorded onto tape in the form of a series of audible signals. These are rectangular in shape, a 3600 Hz tone being followed by one with a frequency of 2400 Hz, and so on (see diagrams (1) and (2) in figure 7). A signal with a frequency of 2400 Hz corresponds to a logic low level while 3600 Hz corresponds to a high logic level.

The circuit around IC6 and IC7 makes sure that the output of IC7 is high when a frequency of 3600 Hz is present at the input connector J1 and that it is low when a frequency of 2400 Hz is present. The RDTAPE subroutine extracts either a logic one or a logic zero from the audio information, depending on the duration of the two tones. Together with the associated components, IC6 constitutes a phase-locked loop (PLL). To explain this type of device in great detail (with complex formulae) would fill an entire book! Therefore, we hope that readers do not object to a relatively brief description of its operation.

Where the interface card is concerned, the PLL may be regarded as a **frequency follower**. This is because it is similar to an emitter follower, where the output voltage imitates the input voltage. In this instance, an internal oscillator alters its frequency to correspond with that of the incoming signal within a certain frequency range and above a minimum input signal level. The internal oscillator produces a signal with a frequency which is proportional to a control voltage (Voltage Controlled Oscillator = VCO). Without any input signal, the free-running frequency of the VCO is approximately 3000 Hz — exactly halfway between the frequencies of the two required audio signals. If the input frequency is 3600 Hz, the VCO frequency will rise by 600 Hz; with an input frequency of 2400 Hz it will drop by 600 Hz. The control voltage has to be increased for the frequency to drop and reduced for the frequency to rise. Thus, the level of the control voltage is directly related to the frequency of the input signal and, after comparing it to a fixed reference voltage, it allows a logical distinction to be made between the two frequencies: logic one for one frequency and logic zero for the other. This principle of operation is often termed 'frequency shift keying' (FSK).

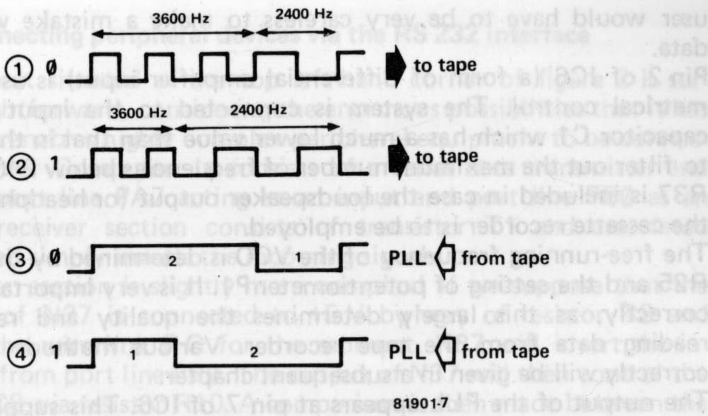


Figure 7. Each data bit which is sent to the cassette tape consists of a high frequency signal followed by a low frequency signal. The duration of each depends on whether the bit is logic one or logic zero. Due to the record/playback chain of the recorder, which acts as a low pass filter, the retrieved signals will not be symmetrical squarewaves. The zero-crossings nevertheless maintain the same position which enables the PLL to produce the compromise product (diagram (3) or (4)), and the software of the TM program deciphers this to give the required output: logic Ø or logic 1.

The power supply for the phase-locked loop is fed via diodes D2 and D3. Capacitor C8 is connected in parallel with the two diodes in order to suppress any transients. This 'clean' supply voltage (about 11 V due to the diodes) is also used to preset the input bias voltages (pins 2 and 3 of IC6) by means of resistors R21 . . . R24. In the KIM computer a 5 volt supply is used and any interference 'spikes' pass straight through to the (in theory) balanced inputs.

Another difference between this system and the KIM is that in the KIM the input signal is not attenuated by a factor of ten until it reaches pin 2. All these improvements make the system so easy to work with that the user would have to be very careless to make a mistake when inputting data.

Pin 2 of IC6 (a form of differential amplifier input) is used as an asymmetrical control. The system is connected to the input socket J1 via capacitor C1 which has a much lower value than that in the KIM in order to filter out the maximum number of frequencies below 2400 Hz. Resistor R37 is included in case the loudspeaker output (or headphone output) of the cassette recorder is to be employed.

The free-running frequency of the VCO is determined by the values of C3, R25 and the setting of potentiometer P1. It is very important to preset P1 correctly, as this largely determines the quality and reliability when reading data from the tape recorder. Various methods for setting P1 correctly will be given in a subsequent chapter.

The output of the PLL appears at pin 7 of IC6. This supplies the control voltage mentioned earlier, to ensure that the VCO output frequency is proportional to the input signal. Capacitor C9, which is connected between the VCO output and +12 V, constitutes the required low pass filter together with an internal 3k6 resistor. This enables the PLL to react quickly to any changes at the output without any fear of 'overshoot'. The output of the PLL is connected to the inverting input of the comparator, IC7, via the ladder filter consisting of resistors R26 . . . R28 and capacitors C5 . . . C7. The non-inverting input of the comparator is connected to a fixed reference voltage produced by IC6 (at pin 6) via resistor R29.

The values of the components in the ladder filter depend on the speed at which the 3600 Hz and 2400 Hz frequencies follow each other. This in turn is determined by the speed at which the data bits are written to and read from the tape. This is commonly called the baud rate — the number of bits transmitted or received per second. In the case of the Junior Computer, the baud rate is 800 (bits per second) for both the hardware and the software.

As you will remember, when the frequency of the VCO rises to 3600 Hz the output voltage at pin 7 of the PLL falls and when the VCO frequency decreases to 2400 Hz the output voltage rises. Therefore, since the filtered DC output from IC6 at the inverting input of the comparator is either higher or lower than the reference voltage present at the non-inverting input, the output of IC7 will go high (logic one) when the input frequency is 3600 Hz and will go low (logic zero) when the input frequency is 2400 Hz. This is exactly what is required. The output voltage of IC7 is then inverted by gate N38 (high goes low and vice versa) before being fed to port line PB7 (see diagrams (3) and (4) in figure 7).

N.B. There is a limit to the speed at which the PLL can react to a change in input frequency. Consequently, the output of IC7 will not change state instantaneously, but will fluctuate between the two logic levels before eventually making a definite choice. This is called PLL 'jitter' and is similar to contact bounce when switches or relays etc. are operated. There is absolutely no need to worry about this phenomenon as the subroutine RDTAPE in the TAPE MONITOR system program takes care of this behaviour.

Connecting peripheral devices via the RS 232 interface

The small circuit situated at the top left-hand corner of figure 2 is surprisingly straightforward, considering the enormous possibilities that it has to offer. For it enables highly complex peripheral equipment to be connected to the Junior Computer. It consists of a serial data transmitter and receiver with port line PA7 acting as an input and port line PBØ as an output. The receiver section consists of transistor T1 and associated components and simply inverts the incoming signal.

The transmitter section is slightly more complex! If we suppose that the input (pin 6) of N37 is connected to +5 V by way of resistor R8 and forget about the wire link P-Q for the moment, N37 will invert the incoming signal from port line PBØ. The output of N37 will then control the comparator, IC8, via resistor R10. A comparison is then made between the output of N37 and the voltage at the junction of R11/R12. When the output of N37 is high the output level of IC8 will be approximately +12 V. Conversely, when the output of N37 is low the output level of IC8 will be about −12 V. Again it can be seen that the output signal from port line PBØ is inverted irrespective of whether the logic levels are adapted to any particular voltage (± 12 V) or not. The data input and output are connected to a standard 25 pin D-type connector, the RS 232 connector. This particular number refers to the universally adopted standard which has been established for data communications. Each data byte is preceded by a single start bit and is followed by one (or sometimes two) stop bits. The data bytes are coded according to the ASCII format.

The RS 232 standard determines the two possible logic levels and their corresponding voltage values. Later on the D-type connector was included in this standard. A logic zero is represented by a voltage between +5 V and +15 V (RS 232C version) and a logic one by a voltage between −5 V and −15 V. In the Junior Computer this corresponds to about +12 V and −12 V respectively. In other words, a low voltage represents a high logic level and vice versa. This is known as negative logic. By inverting the logic levels in the RS 232 interface twice, once during transmission and once during reception, however, the computer does not have to bother about such subtle distinctions. The D-type connector features a number of pins which are internally linked. These links may be altered to suit the particular peripheral device used.

The first suitable device that springs to mind is the Elekterminal, a video display terminal and ASCII keyboard which was originally designed for the Elektor SC/MP system, but which is equally suitable here. The system program PRINTER MONITOR is based on using the ASCII keyboard as an input (through various key commands) and the actual Elekterminal or a suitable printer (not — at least, not for the moment — the metal foil printer published in the March 1980 issue of Elektor magazine) as an output (display).

Now all the hardware involved in the extensions has been discussed, apart from the 'revised' power supply and a few other minor modifications required to the main board of the Junior Computer. Depending on which extensions are to be incorporated, this may involve changing a few resistor values, adding an extra wire link etc. Nevertheless, the power supply for the computer system will have to be modified to cope with the extra current requirements.

The 'revised' main board and power supply

Constructive surgery on the existing boards

Provided a man does not suffer from bad circulation, the heart inside his body should be able to pump blood to the top of his head and to the tip of his toes. Similarly, the power supply for the Junior Computer must be in close contact with the 'extremities' of the machine, the extension boards, at all times. Now that the supply current for the computer (the 'blood'), has to travel such relatively long distances, the 'arteries' of the system will have to be widened to cope with the extra flow. At the same time, the main board at the heart of the computer will have to undergo a little minor surgery in the form of a few modifications here and there. Just as an office worker would have to build up his muscles considerably to be able to tackle the labour of a lumberjack, the Junior Computer has to be 'trained' to withstand the new strain imposed upon it. Again, the amount of 'muscle' that needs to be added all depends on the ambitions of the operator.

Preparing the main board

Actually, the 'operation' involved is not nearly as drastic as it may sound and can be carried out in three steps:

1. **Change one or two resistors.** As you know, the address and data lines have to be buffered before they can serve the various extensions. The control lines, however do not need to be buffered, that is, apart from the RAM-R/W signal. One method of effectively buffering the latter signal is to reduce the value of the pull-up resistor, R5, on the main board connected to it to 470 Ω. The lower resistance value allows the signal to react faster to any change in logic level. The speed of operation of the select signals KØ, K6 and K7 could also be increased by reducing the values of pull-up resistors R14 . . . R16.
2. **Move the wire link** at point D of the address decoder **from ground to point EX.** This modification is more or less optional, but is an absolute must if extra bus board memory is to be added.
3. **Extend the circuitry around gate N5,** in order to disable a non-maskable interrupt (NMI) on specific occasions and thus prevent a program from being run in the STEP mode. Figure 8a presents the situation 'as of old', whereas figure 8b shows the modified version. Figure 9, on the other hand, gives the pulse diagrams for the various signals around the circuit.

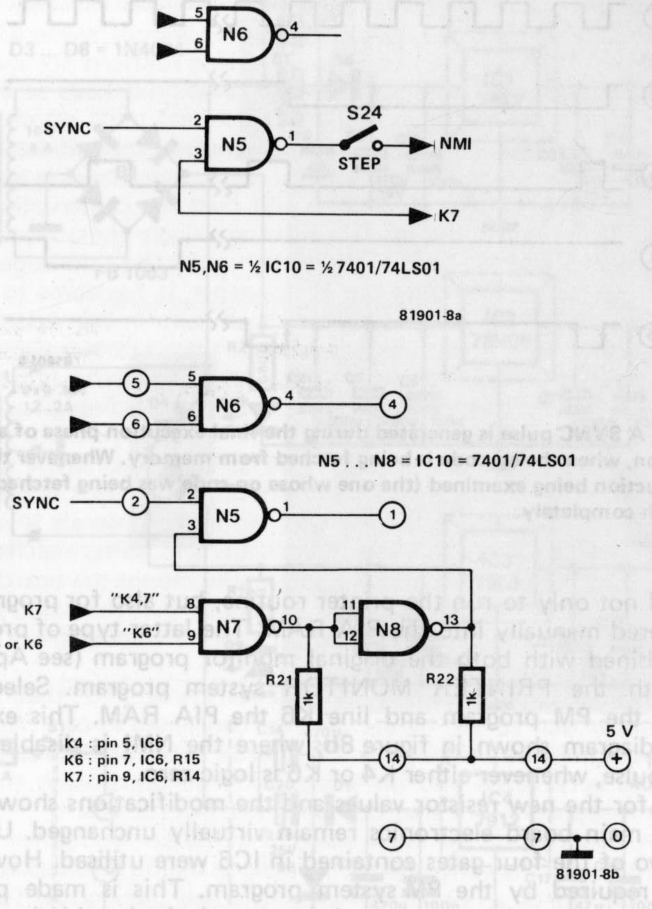


Figure 8. The additional circuitry required on the main board to inhibit the step function (by disabling the NMI). This is necessary when the PM program or the original monitor routine is called (see Appendix 2).

Those of you who read Book 1 will remember from the instruction listing at the back of the book that every instruction takes a certain amount of time to be executed. This is expressed as a whole number of clock periods. During one clock period the op-code of the next instruction to be executed is fetched from memory. This occurs during the last phase of the instruction currently on display. Every time data is fetched from memory, the R/W signal (see diagram 2 in figure 9) has to become logic one (after all, an op-code has to be read!) and a SYNC pulse will cause an interrupt (NMI) by way of gate N5, **unless the K7 select line goes low.** This means that the NMI will be enabled unless the EPROM on the main board of the Junior Computer is being accessed. When a non-maskable interrupt occurs, execution of the current instruction (the one whose op-code was fetched) will be completed.

In the STEP mode, the NMI jump vector will be pointing to the start address of the monitor program (1CØØ), meaning that as soon as the computer encounters monitor instructions, the NMI will be disabled and the machine will leave the step mode. This is quite logical, seeing as the whole purpose of the monitor program is to run through a complete series of instructions **without interruption.** This happens, for instance, when the computer is waiting for a new key to be depressed and when it multiplexes the six displays.

When we get to chapter 12, it will be seen that the NMI also needs to be

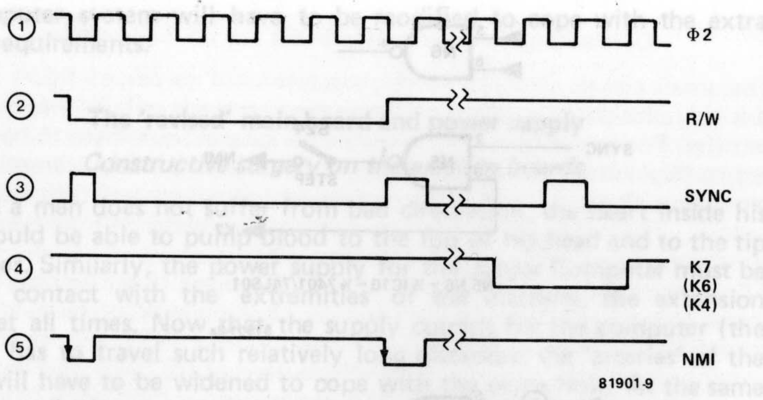


Figure 9. A SYNC pulse is generated during the final execution phase of an instruction, when the op-code is being fetched from memory. Whenever this happens, the instruction being examined (the one whose op-code was being fetched) will be dealt with completely.

disabled not only to run the printer routine, but also for programs which are entered manually into the PIA RAM. The latter type of program may be combined with both the original monitor program (see Appendix 2), and with the PRINTER MONITOR system program. Select line K4 enables the PM program and line K6 the PIA RAM. This explains the circuit diagram shown in figure 8b, where the NMI is disabled during a SYNC pulse, whenever either K4 or K6 is logic zero.

Except for the new resistor values and the modifications shown in figure 8b, the main board electronics remain virtually unchanged. Up to now, only two of the four gates contained in IC6 were utilised. However, they are all required by the PM system program. This is made possible by adding a tiny module to the existing board. Again, this is an optional 'extra' for readers who wish to use a printer in the step mode. Another point about the module is that it allows decimal calculations to be carried out without any problems, even if the TM program is not used (see Appendix 2).

Boosting the power supply

More (and larger) 'mouths' to feed . . .

'Feeding the five thousand' does not apply in electronics. There may well be enough power available for two boards, but not necessarily for three (or more)! This is especially true where EPROMs are concerned, as they are particularly greedy components. In addition, a −12 V supply is required for the RS 232 interface (see figure 2).

The circuit diagram of the modified power supply is shown in figure 10. It produces:

+5 V, 4 A maximum (previously 1 A)
−5 V, 400 mA maximum (previously 100 mA)

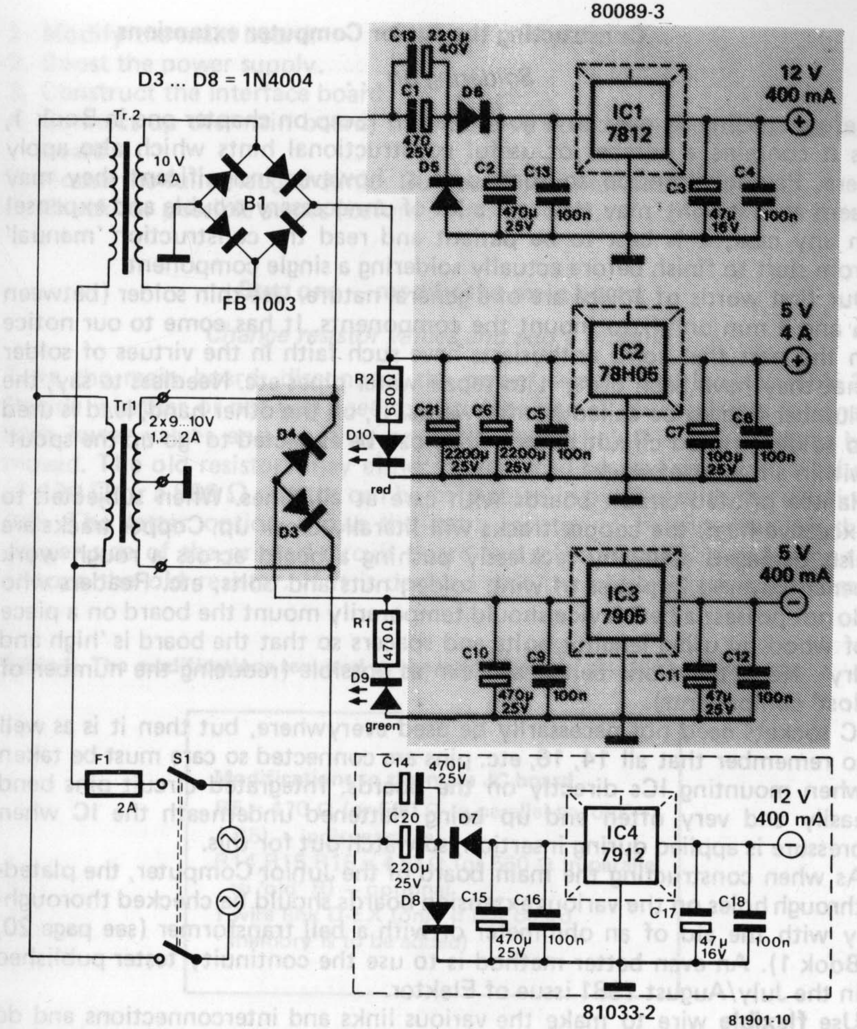


Figure 10. The circuit diagram of the 'revised' Junior Computer power supply.

+12 V, 400 mA maximum (previously 100 mA)
−12 V, 400 mA maximum (new)

This should be more than enough to supply the main board, the interface board and up to five additional memory cards. The Elekterminal has its own built-in power supply, but it can also be powered by the revised supply if required. How this can be accomplished will be described in chapter 12.

That just about covers the extensions to the Junior Computer (apart from the bus boards) at least on paper. The RAM/EPROM card will be considered in the third and final section of this chapter. First of all, it is high time the theory was put into practice. At last we can get on with the various constructional details!

Constructing the Junior Computer extensions

Soldering on . . .

Before starting, it may be a good idea to recap on chapter one in Book 1, as it contains a number of useful constructional hints which also apply here. Paying attention to such details, however insignificant they may seem at first sight, may well save a lot of unnecessary trouble and expense! In any case, it is best to be patient and read the construction 'manual' from start to finish before actually soldering a single component.

Our first words of advice are of a general nature. Use thin solder (between ½ and 1 mm only!) to mount the components. It has come to our notice in the past that some enthusiasts have such faith in the virtues of solder that they have been using it to repair water pipes etc. Needless to say, the plumber had to be called in afterwards. If, on the other hand, lead is used to solder printed circuit boards, they can be expected to 'go up the spout' within a matter of weeks.

Handle printed circuit boards with care at all times. When subjected to excessive heat, the copper tracks will literally crack up. Copper tracks are also damaged easily by recklessly pushing a board across a rough work bench covered in pieces of wire, solder, nuts and bolts, etc. Readers who do not possess a bench vice should temporarily mount the board on a piece of wood by using lengthy bolts and spacers so that the board is 'high and dry'. Keep the work bench as clean as possible (reducing the number of 'lost' components).

IC sockets need not necessarily be used everywhere, but then it is as well to remember that all 14, 16, etc. pins are connected so care must be taken when mounting ICs directly on the boards. Integrated circuit pins bend easily and very often end up being flattened underneath the IC when pressure is applied during insertion, so watch out for this.

As when constructing the main board of the Junior Computer, the plated-through holes on the various extension boards should be checked thoroughly with the aid of an ohmmeter or with a bell transformer (see page 20, Book 1). An even better method is to use the continuity tester published in the July/August 1981 issue of Elektor.

Use **flexible** wire to make the various links and interconnections and do not be stingy about the diameter. Flexible wire consists of many strands and the links should be at least 0.93 mm in diameter. **The main power supply connection leads should have a diameter of not less than 1 mm!**

Callers who ring up the Elektor Technical Queries department on Monday afternoons often admit, somewhat reluctantly, that they throw in the towel a little too easily. As soon as something goes wrong, or does not work straight away, they rush to their local component shop to buy a few more ICs. **This is totally unnecessary 99% of the time!** It is much wiser, and certainly cheaper, to make sure that all the connections and polarities are correct before actually turning on the power supply, thereby avoiding short-circuits etc.

After that little lecture, which unfortunately often proves to be indispensible, we can continue with the main task at hand: developing the Junior Computer to maturity. Each item is dealt with separately in a series of steps which should be followed closely in the order specified:

1. Modify the main board.
2. Boost the power supply.
3. Construct the interface board.
4. Connect up the main board, the interface board and the (optional) bus board.
5. House the fully-fledged Junior Computer in a new case.
6. Check for possible errors (before switching on!).

Step one — modify the main board

Change resistor values and add a module

Take the main board, disconnect the power supply and look at table 3. Decide whether or not you need to exchange one or four resistors for ones with lower values and whether or not the wire link D — EX needs to be moved. The old resistors may either be replaced by new ones with a value of 470 Ω or a 560 Ω resistor can be connected in parallel with the existing one. The latter option allows the extra resistor to be mounted on the copper side of the printed circuit board and saves the trouble of having to remove the old resistor. If it is decided that the old resistors must come out, a good quality 'solder-sucker' should be used. Alternatively, the leads of the resistor can be cut at both ends and the bits of wire remaining can be removed with a pair of needle-nose pliers after applying a little heat to the solder joints with the soldering iron.

Table 3. The modifications required to the main board of the Junior Computer.

Modifications to the main JC board

R5 = 470 Ω (or 560 Ω in parallel to 'old' R5) — indispensable

R14,R15,R16 = 470 Ω (or 560 Ω in parallel to 'old' R) — optional

1 wire link D-EX (only if bus board memory is to be added)

The colour-codes for the resistor values used are as follows:

470 Ω: yellow - violet - brown (- gold)

560 Ω: green - blue - brown (- gold)

If extra (bus board) memory is to be added, the wire link at point D of the address decoder, IC6, must be moved. Point D must now be connected to point EX.

Readers who do not intend to utilise the PRINTER MONITOR system program and its single-step facility (they don't know what they are missing!) can skip the rest of this section and move directly on to step two. For the rest (most?) of us it is now time to mount the **module**:

The circuit diagram for the complete module is shown in figure 8b; the printed circuit board and component overlay is provided in figure 11; the wiring details are given in figure 12 and the components required are listed in table 4.

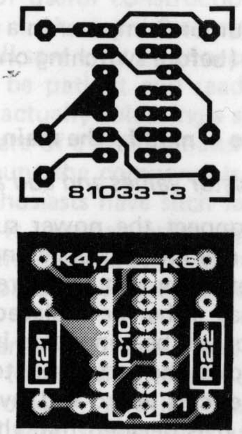


Figure 11. The module containing the circuit shown in figure 8b. Six of the seven pins belonging to IC10 which are to be connected to the main board are indicated by crosses inside a circle. Unfortunately, the seventh (pin 14) is unmarked!

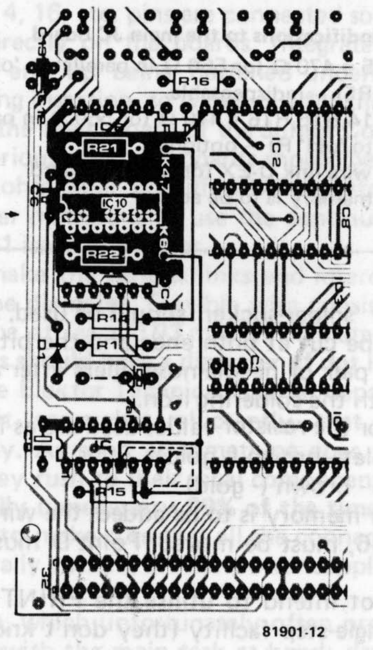


Figure 12. This is how the module shown in figure 11 is connected to the main board. Two wire links are involved: one leading to K6 and one to either K4 or K6.

Table 4. The parts list for the module: EPS 81033-3.

module EPS 81033-3

R21,R22 = 1 k: brown-black-red-(gold)
(continuation of numbers on main JC board)
IC10 = 7401, 74LS01 (only if original IC10 has to be desoldered)
1 printed circuit boad EPS 81033-3
1 14 pin IC socket (provided IC10 was not already in socket)
2 solder pins

Use is now made of all four gates inside IC10, two of which were previously unused: N7 and N8. The module contains a substitute for IC10, two resistors and two solder pins to connect the required select lines (the SYNC and K7 signals are not included on the expansion connector, therefore the interface board cannot be used for this particular application). The supplementary circuit is mounted 'piggy-back' on top of the main board.

There will be no problems here if IC10 was originally mounted in a socket. If not, a socket will have to be installed after first removing the original IC by careful application of a pencil point soldering iron and a good quality 'solder-sucker'. Alternatively, the pins of the IC can be snipped off and removed with a pair of needle-nose pliers. Operate with care and a steady hand!

On the copper track side of the module board only pins 1, 2, 4 . . . 7 and 14 will be used. These pins act as links with the socket which has just been mounted on the main board in place of IC10 and are indicated on the component overlay of the module by seven circles with crosses in them. Pins 3 and 8 . . . 13 are not used (they are interconnected on the module only) and can be cut off. As an alternative, seven short lengths of wire may be used to link the module directly to the main board.

Now the select (K) lines can be dealt with. In some instances the wiring can be connected directly to the lead of a resistor (see figure 12). If the PM program is required, lines K4 (together with line K5, as will become apparent) and K6 will be connected to the module. (If not, lines K6 and K7 are connected to IC10 instead).

Step two – boost the power supply

How to get a few more (milli) amps . . . and minus 12 V

The circuit diagram of the modified power supply is shown in figure 10 and the printed circuit board and component layout for the −12 V section is given in figure 13. The printed circuit board for the existing power supply is shown in chapter 1, Bok 1 pages 29 and 30. The new parts list is printed in table 5 and a constructional drawing is provided in figure 14.

To start with, diodes D1 and D2 are removed from the existing power

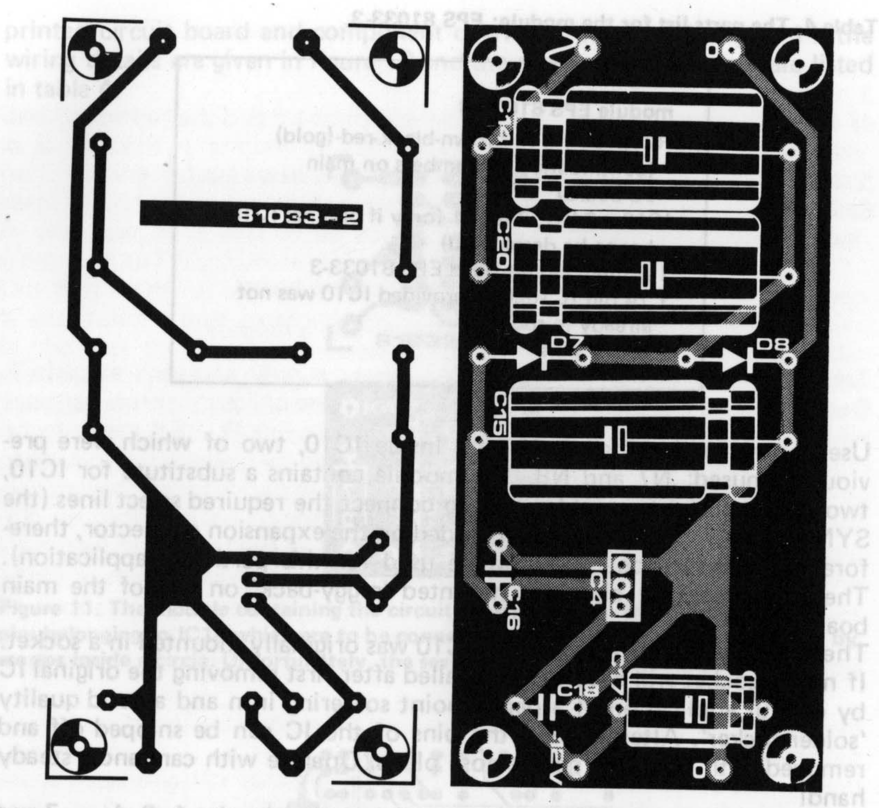
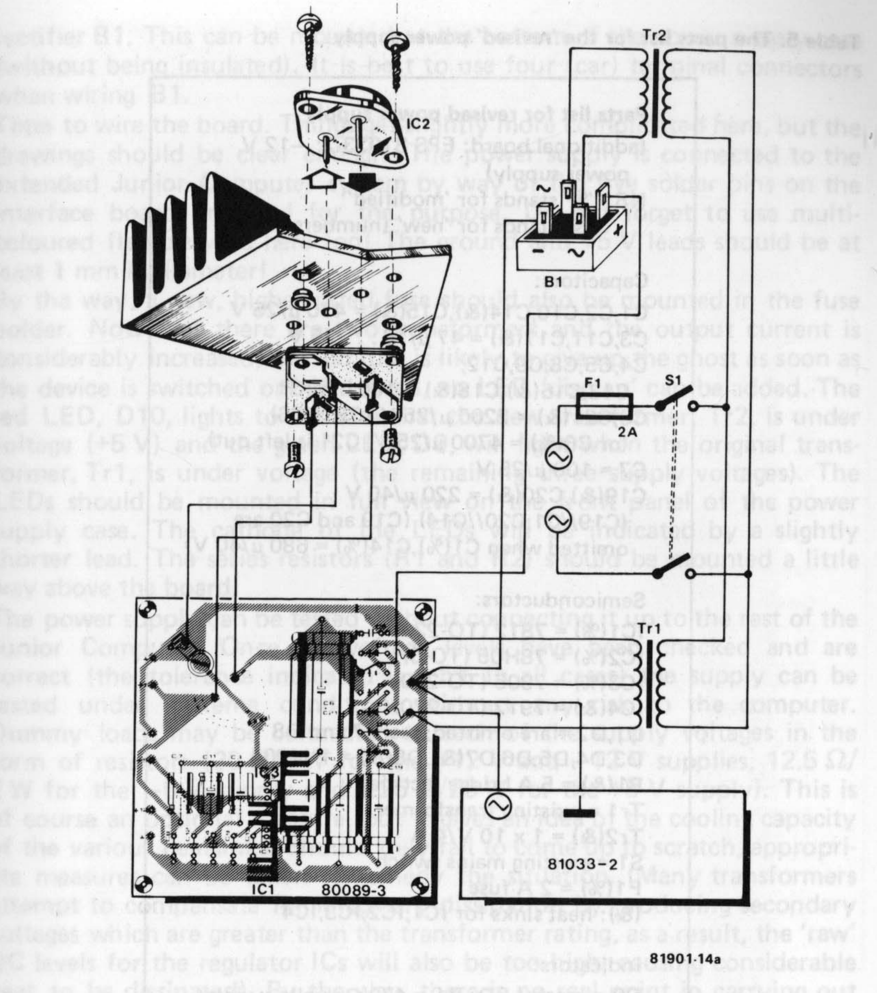


Figure 13. The printed circuit board and component overlay for the −12 V power supply.

supply board (they can now be used for diodes D7 and D8), so are the voltage regulators IC1 . . . IC3 along with the heatsink. After virtually demolishing the board it is time to re-build it. Capacitor C19 is connected in parallel to and above C1. It is also possible to replace C1 by a 680 μF/40 V electrolytic capacitor. Likewise, C21 is mounted in parallel with C6. Again, C6 may be substituted for a 4700 μF/25 V electrolytic. It all depends on what happens to be available.

Next, the new regulator ICs (IC1 and IC3) are introduced to replace their predecessors and are mounted on the board. Read and re-read the following very carefully: **the metal face of each IC is situated to one side of C2.** In other words, disregard the component overlay! The two ICs should be provided with a suitable heatsink, as shown in figure 14b. The centre pin of each is linked internally to the metal face and therefore to the heatsink. It is quite obvious from the pin assignments given in figure 14b that **the heatsinks of IC1 and IC3 must not touch.** The solution is to bend the pins of one of the ICs while keeping them vertical and then mount the two heatsinks so that they face in opposite directions.

The new regulator which takes the place of the original IC2 is mounted on a pre-drilled TO-3 heatsink which can be mounted on the rear of the case.



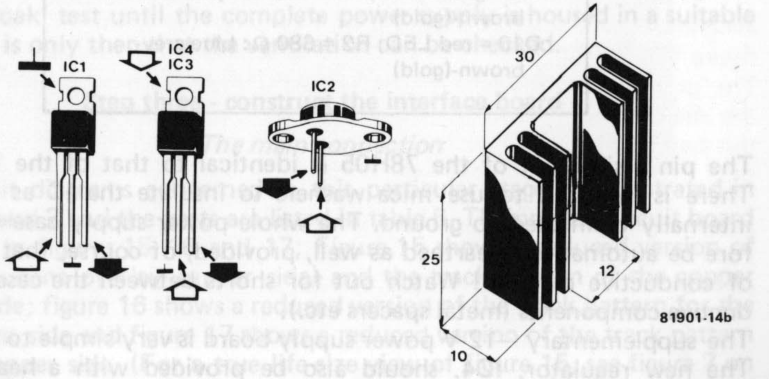


Figure 14. The most important details concerning the installation and wiring of the revised power supply (14a) and a few items concerning the voltage regulator ICs and heatsinks.

Table 5. The parts list for the 'revised' power supply.

Parts list for revised power supply
(additional board: EPS 81033-2: −12 V
power supply)
NB. '%' stands for 'modified'
'&' stands for 'new' (numbers run on)

Capacitors:
C1,C2,C10,C14(&),C15(&) = 470 μ/25 V
C3,C11,C17(&) = 47 μ/16 . . . 25 V
C4,C5,C8,C9,C12,
C13,C16(&),C18(&) = 100 n MKH
C6,C21(&) = 2200 μ/25 V (C21//C6)
(or C6(%) = 4700 μ/25 V; C21 is left out)
C7 = 100 μ/25 V
C19(&),C20(&) = 220 μ/40 V
(C19//C1;C20//C14) (C19 and C20 are
omitted when C1(%),C14(%) = 680 μ/40 V)

Semiconductors:
IC1(%) = 7812 (TO-220)
IC2(%) = 78H05 (TO-3)
IC3(%) = 7905 (TO-220)
IC4(&) = 7912 (TO-220)
D1,D2 = are omitted; see D7 and D8
D3,D4,D5,D6,D7(&),D8(&) = 1N4004
B1(&) = 5 A bridge rectifier
Tr1 = existing transformer
Tr2(&) = 1 x 10 V/4 A
S1 = existing mains switch
F1(%) = 2 A fuse
(&): heat sinks for IC1,IC2,IC3,IC4

Indicators:
D9 = green LED; R1 = 470 Ω: yellow-violet-
brown-(gold)
D10 = red LED; R2 = 680 Ω: blue-grey-
brown-(gold)

The pin assignment of the 78H05 is identical to that of the LM 309K. There is no need to use mica washers to insulate the IC as its case is internally connected to ground. The whole power supply case will therefore be automatically earthed as well, provided, of course, that it is made of conductive material! Watch out for shorts between the case and conductive components (metal spacers etc.).

The supplementary −12 V power supply board is very simple to construct. The new regulator, IC4, should also be provided with a heatsink (see figure 13). An additional transformer is required for the −12 V supply. When wiring up the power supply, the drawing in figure 14a should help enormously. It shows how to connect another newcomer, the bridge rectifier B1. This can be mounted at the bottom of the power supply case (without being insulated). It is best to use four (car) terminal connectors when wiring B1.

Time to wire the board. Things get slightly more complicated here, but the drawings should be clear enough. The power supply is connected to the extended Junior Computer system by way of the five solder pins on the interface board provided for the purpose. Do not forget to use multicoloured flexible wire here too! The ground and +5 V leads should be at least 1 mm in diameter!

By the way, a new, higher rated fuse should also be mounted in the fuse holder. Now that there are two transformers and the output current is considerably increased, the old fuse is likely to give up the ghost as soon as the device is switched on! If desired, an LED 'display' can be added. The red LED, D10, lights to indicate that the new transformer, Tr2, is under voltage (+5 V) and the green LED, D9, will light when the original transformer, Tr1, is under voltage (the remaining three supply voltages). The LEDs should be mounted in full view on the front panel of the power supply case. The cathode of the LEDs will be indicated by a slightly shorter lead. The series resistors (R1 and R2) should be mounted a little way above the board.

The power supply can be tested without connecting it up to the rest of the Junior Computer. Once the voltage levels have been checked and are correct (the tolerance indicated is ± 5% in all cases) the supply can be tested under extreme conditions without any risk to the computer. Dummy loads may be connected to each of the supply voltages in the form of resistors (30 Ω/5 W for the +12 V and −12 V supplies, 12.5 Ω/2 W for the −5 V supply and 1.25 Ω/20 W for the +5 V supply). This is of course an optional measure, but it gives an idea of the cooling capacity of the various heatsinks. Should they fail to come up to scratch, appropriate measures can be taken to remedy the situation. (Many transformers attempt to compensate for imaginary dissipation by producing secondary voltages which are greater than the transformer rating, as a result, the 'raw' DC levels for the regulator ICs will also be too high, causing considerable heat to be dissipated). By the way, there is no real point in carrying out such a 'soak' test until the complete power supply is housed in a suitable case, as it is only then that the ventilation can be checked.

Step three - construct the interface board

The main connection

The circuit diagrams concerned in this particular stage are illustrated in figures 1 and 2 and the parts are listed in table 6. The printed circuit board is shown in figures 15, 16 and 17; Figure 15 shows a reduced version of the component overlay (upper side) and the track pattern of the copper (lower) side; figure 16 shows a reduced version of the track pattern for the component side and figure 17 shows a reduced version of the track pattern for the copper side. (For a true life-size view of figure 15, see figure 7 on page 6-14 of the June 1981 issue of Elektor).

The wire links for IC4 and IC5 must be installed according to table 7 and details concerning the connectors are presented in figure 19. The pin

Table 6. The parts list for the interface board.

Parts list for the interface board

Resistors:
R1,R2,R3,R4,R32,R33,R34,R35 = 1 k: brown-black-red-(gold)
R5 = 22 k: red-red-orange-(gold)
R6,R10,R11,R14,R15,R24,R26,R27, R28 = 10 k: brown-black-orange-(gold)
R7,R8,R36 = 8k2: grey-red-red-(gold)
R9,R18,R22,R23 = 4k7: yellow-violet-red-(gold)
R12 = 6k8: blue-grey-red-(gold)
R13,R25,R31 = 2k2: red-red-red-(gold)
R16 = 100 Ω: brown-black-brown-(gold)
R17 = 330 Ω: orange-orange-brown-(gold)
R19 = 470 Ω: yellow-violet-brown-(gold)
R20 = 1k2: brown-red-red-(gold)
R21 = 15 k: brown-green-orange-(gold)
R29 = 33k: orange-orange-orange-(gold)
R30 = 4M7: yellow-violet-green-(gold)
R37 = 33 Ω (see text): orange-orange-black (gold)
P1 = 5 k (4k7) multiturn preset
P2 = 1 k preset

Capacitors:
C1 = 220 n MKH
C2,C11,C12,C13 = 10 μ/16 V tantalum
C3 = 22 n MKH
C4 = 1 n MKH
C5,C6,C7 = 6n8 MKH
C8 = 100 n MKH
C9 = 47 n MKH
C10,C14 . . . C22 = 1 μ/16 V tantalum (total 10)

Semiconductors:
T1 = BC 547B
T2,T3 = BC 516
D1,D2,D3 = 1N4148
D4 = LED green
D5 = LED red
IC1 = 6522 (Rockwell, Synertek)
IC2,IC3 = 2114
IC4 = 2716, 2708, 8114
IC5 = 2716, 2708, 8114
IC6 = 565
IC7,IC8 = 311
IC9,IC10 = 74LS241
IC11,IC12 = 74LS243
IC13 = 74LS27, 7427
IC14 = 74LS01, 7401
IC15 = 74LS30, 7430
IC16 = 74LS00, 7400
IC17 = 82S23, 74188

Miscellaneous:
Re1,Re2 = DIL reed relays
2 8-pin IC sockets
9 14-pin IC sockets
1 16-pin IC socket
2 18-pin IC sockets
2 20-pin IC sockets
2 24-pin IC sockets (see text)
1 40-pin IC socket
5 wire links on board (in addition to ones marked alphabetically)
J1 . . . J4 = cinch chassis connectors
1 25-pole D connector (RS 232), mounted at right angles to board (see figure 19e)
20 solder pins (VIA 'connector')
29 solder pins (marked A, B, C, etc.)
1 input connector (64-pin) placed at right angles, DIN 41612, male! (is identical to expansion connector in standard JC) — see figure 19a
5 solder pins (links to port connector)
5 solder pins (links to power supply)
3 solder pins (LED connections)
1 output connector (64-pin) placed at right angles, female (see figure 19c and text)

assignments for all the ICs used on the interface board are provided in figure 18.
The printed circuit board is double-sided with plated-through holes like the main board. There is, however, one fundamental difference between the two boards: the interface board has a component overlay on one side only. This does not mean that all the components are mounted on that side. Most of the connectors will in fact be placed on the copper side. Generally speaking, the component overlay is considered to be situated on the upper side of the board. Well, this tradition will have to be broken with as we are going to make a 'sandwich' with the lower slice being the interface board and the upper slice the main board.
Several readers may have expressed a certain amount of surprise (and confusion!) when first examining figure 15, which shows the component overlay of one side and the copper track pattern of the **other** side. As you will have probably gathered by now, this is all to do with the way it has to be printed. The track pattern corresponding to the copper side is, of

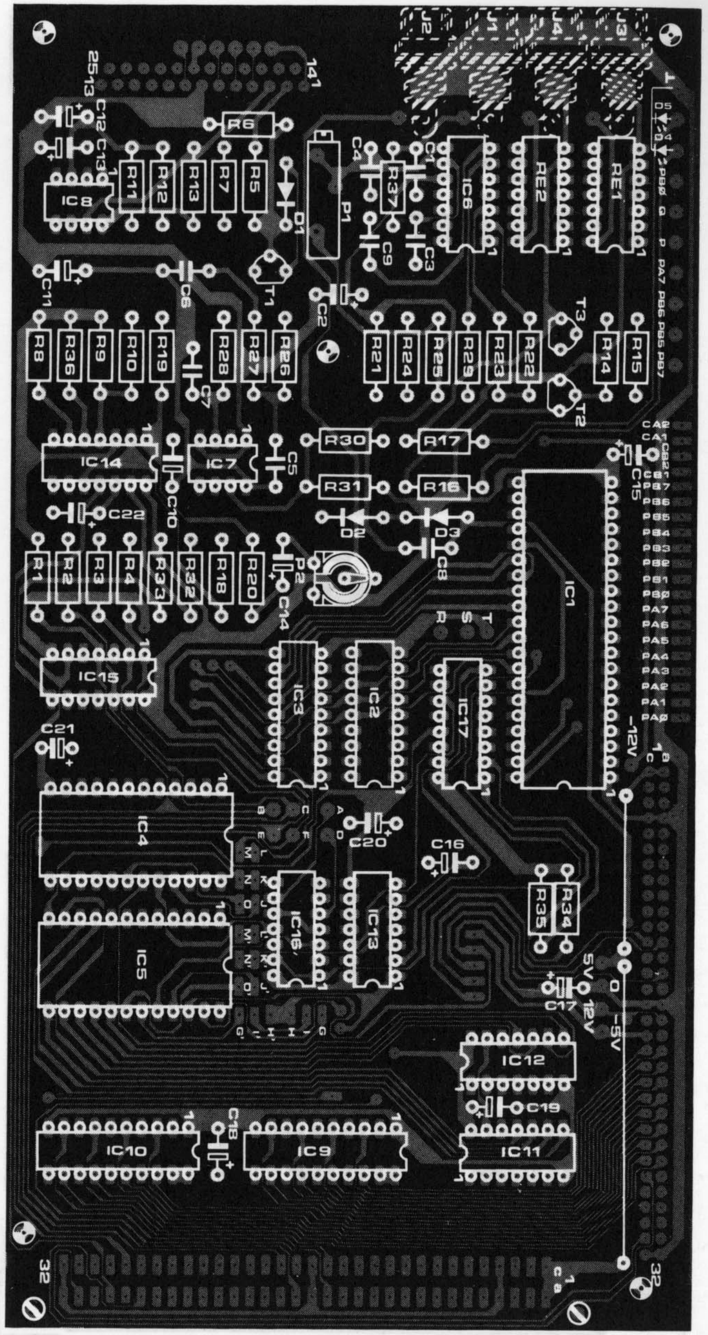


Figure 15. The component overlay of the interface board, including the track pattern of the 'copper' side. For reasons of space, the board is shown reduced here.

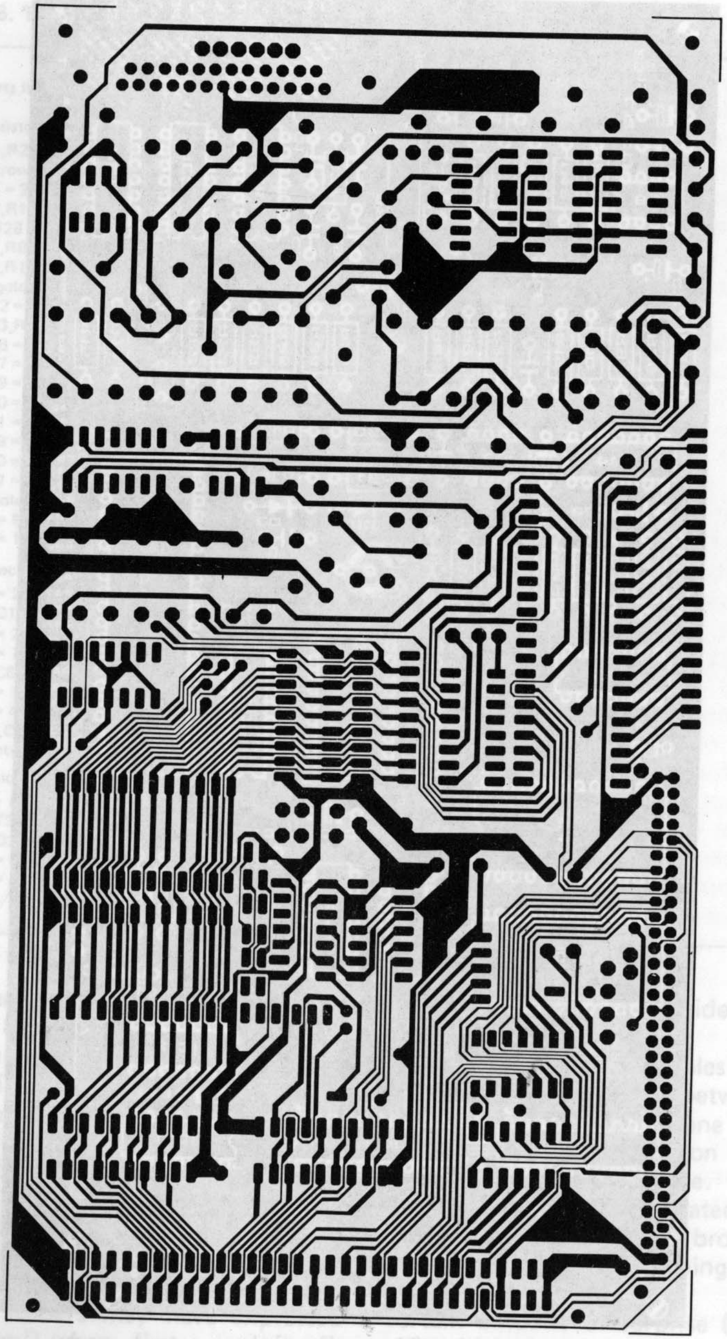


Figure 16. The track pattern of the component side of the interface board (reduced).

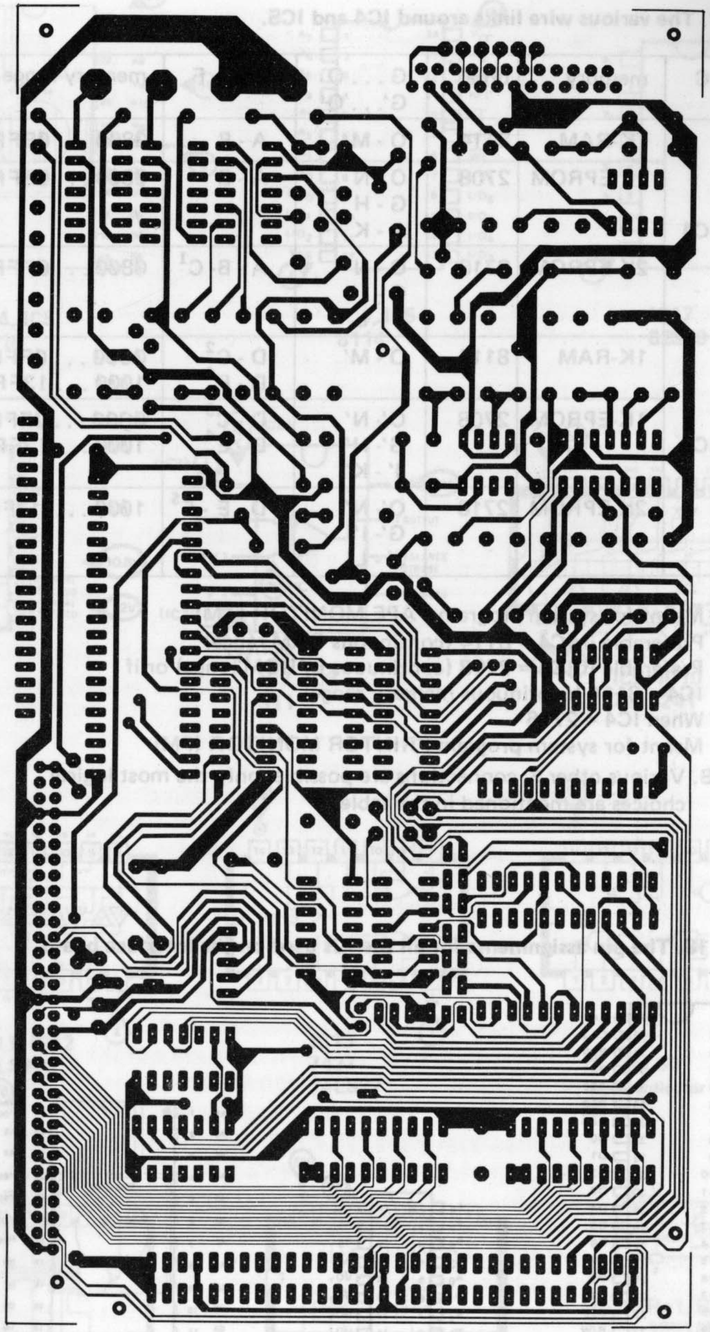


Figure 17. The track pattern of the copper side of the interface board (reduced).

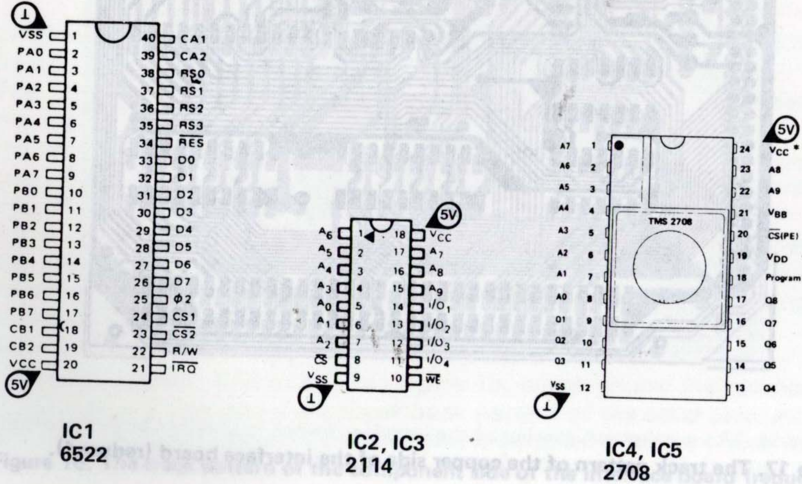
Table 7. The various wire links around IC4 and IC5.

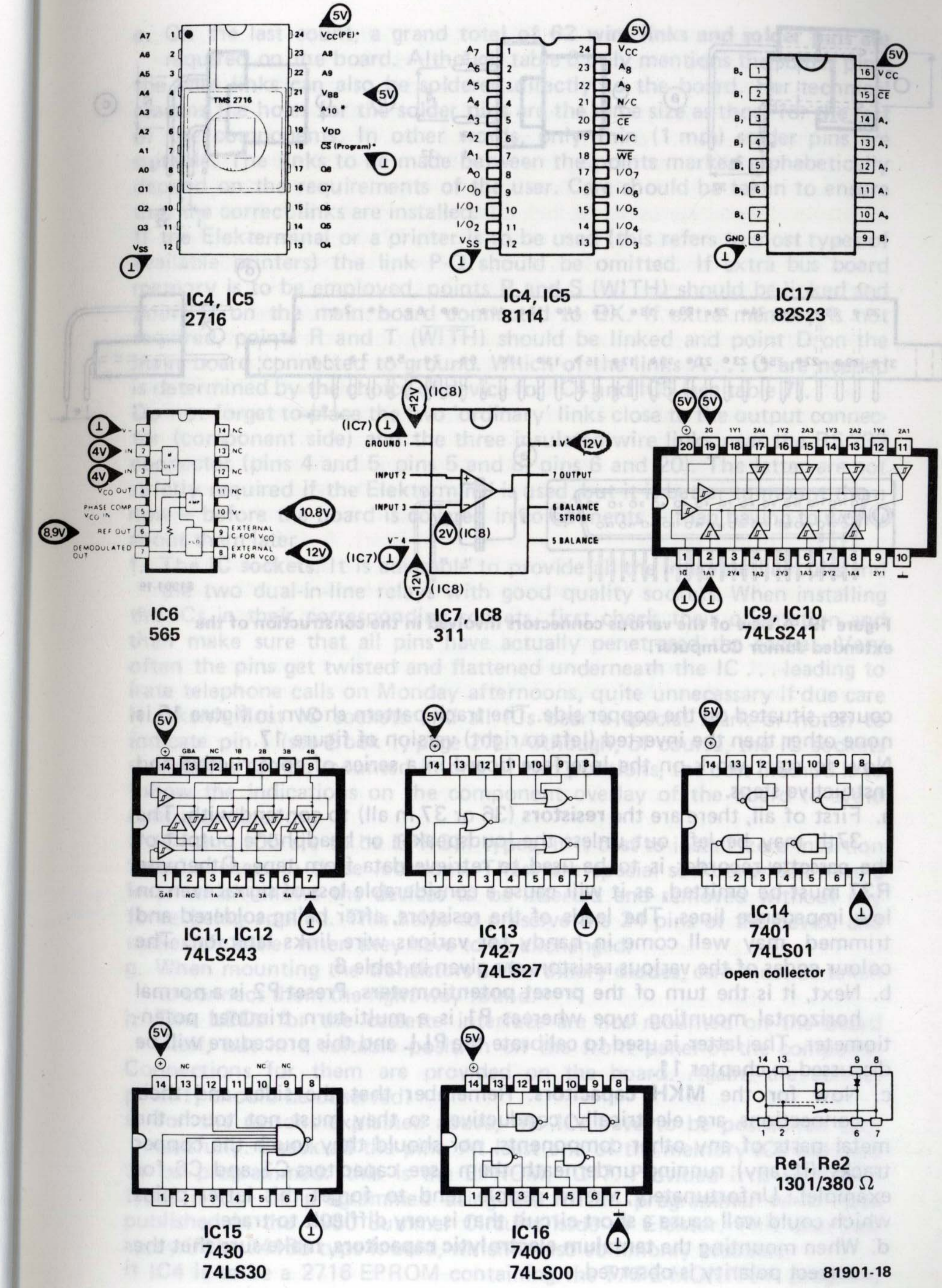
IC	memory	type	G . . . O G'. . . O'	A . . . F	memory range
IC4	1K-RAM	8114	O - M	A - B	Ø8ØØ . . . ØBFF
	1K-EPROM	2708	O - N G - H J - K	A - B	Ø8ØØ . . . ØBFF
	2K-EPROM	2716	O - N G - I J - L	A - B- C^1	Ø8ØØ . . . ØFFF^1
IC5	1K-RAM	8114	O'- M'	D - C^2 D - E^4	ØCØØ . . . ØFFF^2 1ØØØ . . . 13FF^4
	1K-EPROM	2708	O'- N' G'- H' J' - K'	D - C^3 D - E^4	ØCØØ . . . ØFFF^3 1ØØØ . . . 13FF^4
	2K-EPROM	2716	O'- N' G'- I' J' - L'	D - E - F^5	1ØØØ . . . 17FF^5

1) Meant for system program TAPE MONITOR (TM)
2) Preferable if IC4 = 8114 (continuous RAM range)
3) Preferable if IC4 = 2708 (continuous EPROM range) or if IC4 = 8114 (continuous memory range)
4) When IC4 = 2716
5) Meant for system program PRINTER MONITOR (PM)

N.B. Various other K connections are possible, only the most logical choices are mentioned in the table.

Figure 18. The pin assignments for all the ICs used on the interface board.





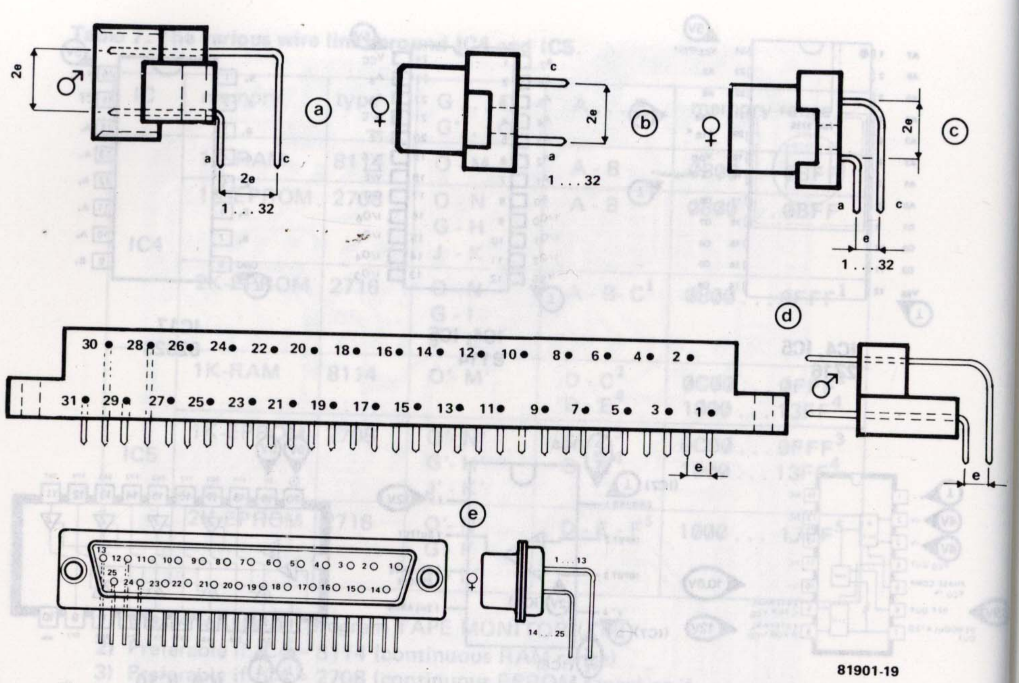


Figure 19. A view of the various connectors involved in the construction of the extended Junior Computer.

course, situated on the copper side. The track pattern shown in figure 15 is none other than the inverted (left to right) version of figure 17.

Now to start work on the interface board in a series of constructive and instructive steps.

a. First of all, there are the **resistors** (36 or 37 in all) to contend with. The 37th may be left out unless the loudspeaker or headphone output of the cassette recorder is to be used to retrieve data from tape. Otherwise R37 must be omitted, as it will cause a considerable loss of signal even on low impedance lines. The leads of the resistors, after being soldered and trimmed, may well come in handy for various wire links later on. The colour codes of the various resistors are given in table 6.

b. Next, it is the turn of the **preset potentiometers**. Preset P2 is a normal horizontal mounting type whereas P1 is a multi-turn trimmer potentiometer. The latter is used to calibrate the PLL and this procedure will be discussed in chapter 11.

c. Now for the **MKH capacitors**. Remember that the sides and their connections are electrically conductive, so they must not touch the metal parts of any other components, nor should they touch the copper tracks (if any) running underneath them (see capacitors C1 and C6 for example). Unfortunately, many people tend to forget this latter point which could well cause a short-circuit that is very difficult to trace.

d. When mounting the **tantalum electrolytic capacitors**, make sure that the correct polarity is observed.

e. On the last count, a grand total of **62 wire links and solder pins** are required on the board. Although table 6 only mentions the solder pins, the wire links can also be soldered directly to the board. For technical reasons the holes for the solder pins are the same size as those for the rest of the components. In other words, only thin (1 mm) solder pins are suitable. The links to be made between the points marked alphabetically depend on the requirements of the user. Care should be taken to ensure that the correct links are installed.

If the Elekterminal or a printer is to be used (this refers to most types of available printers) the link P-Q should be omitted. If extra bus board memory is to be employed, points R and S (WITH) should be linked and point D on the main board connected to EX. If extra memory is not required, points R and T ($\overline{\text{WITH}}$) should be linked and point D on the main board connected to ground. Which of the links A . . . O are needed is determined by the choice of device for IC4 and IC5 (see table 7).

Do not forget to place the two 'ordinary' links close to the output connector (component side) and the three insulated wire links near the RS 232 connector (pins 4 and 5, pins 5 and 8, pins 6 and 20). The latter are not strictly required if the Elekterminal is used, but it is better to mount them now — before the board is covered in components — than having to worry about them later.

f. The **IC sockets**. It is advisable to provide all the integrated circuits and the two dual-in-line relays with good quality sockets. When installing the ICs in their corresponding sockets, first check their orientation and then make sure that all pins have actually penetrated the socket. Very often the pins get twisted and flattened underneath the IC . . . leading to irate telephone calls on Monday afternoons, quite unnecessary if due care is taken! Most IC sockets and all ICs bear a special mark or notch to indicate pin 1 (see Book 1, page 27). Although, of course, the IC sockets themselves can be mounted in one of two positions, the best method is to follow the indications on the component overlay of the board to avoid confusion later on.

If IC4 and IC5 are to be EPROM types, it is best to install 'zero insertion force' sockets in these locations. These are special sockets which have a lever which allows the devices to be inserted and removed without any force being applied. This helps to preserve the 24 pins of the device and makes life easier when they have to be exchanged.

g. When mounting the **transistors and ordinary diodes**, care must be taken to connect them the right way round.

h. The **LEDs** for the cassette interface are not mounted on the board itself, but in a suitable position on the front panel of the computer. Connections for them are provided on the board. Again, the correct polarity should be observed!

i. For the reasons explained above, the **ICs** have to be positioned very carefully. Check **all** the pins. At least one of the memory ICs will have to be programmed: this is the EPROM, IC17. Provided it is an 82S23 type, it can be programmed using the PROM programmer which was published in the 1980 Summer Circuits issue of Elektor. This does not apply if the 74188 type is used, which is also commonly available.

If IC4 is to be a 2716 EPROM containing the TAPE MONITOR program, this can be obtained from Technomatic Ltd, 17 Burnley Road, London NW10. Alternatively, readers can program the device themselves with the aid of the hex dump given at the end of this book. If the TM option is not required, all the cassette hardware may just as well be omitted, unless readers have their own personal cassette routines. The same is true of IC5. If it is to be a 2716 containing the PRINTER MONITOR program it can once again be obtained from Technomatic. Again, readers may program the device themselves using the hex dump at the back of the book (on the understanding, of course, that readers own 2716 programming equipment!). If the G and S functions of the PM program are required, the TM routine must also be available.

j. **Relays** Re1 and Re2, their corresponding sockets and connectors J3 and J4, are rendered superfluous if software control of the cassette decks is deemed unnecessary. In fact, the relays are only really useful when two cassette recorders are used: one to record/store programs (OUTPUT) and one to retrieve data (INPUT). In addition, the two cassette decks should have remote control facilities.

k. **Chassis connectors J1 . . . J4** link the cassette recorder(s) to the Junior Computer. They can be mounted directly on the interface board (on the copper side), with suitably positioned holes in the right-hand side of the case (further details of this will be given later). A different method, and a simpler one, is to mount the sockets remotely at the rear of the case. Connectors J3 and J4, if required (see j), have to be insulated, that is, they must not be connected to ground. Screened leads should be used for the connections to the cassette recorder(s).

l. **The input connector** (if required – see step 4) is mounted on the component side of the board. This is situated on the right-hand edge of the board if it is held so that the EPS number can be read easily! The procedure is the same as that for the IC sockets (see point f). It may well be advisable to have a look at figure 19a at the same time.

m. **The RS 232 connector.** The 25 pin D-type connector that should be used is described in table 6 and is illustrated in figure 19e. If this connector is to be mounted on the (copper side of the) board, at the opposite end to the input connector, the pins should be at right-angles to the main body. Mounting it on the copper side of the interface board has the added advantage that the space between the two boards is used as economically and as efficiently as possible, preventing the sandwich from becoming a king-size hamburger! If the RS 232 connector is mounted on the board, the peripheral devices will be connected up to the right-hand side of the case. If, on the other hand, the connector is placed on the back of the case (use a normal type which can be linked with wires), it can be connected to the board with 'ribbon' cable.

n. **The VIA 'connector'** comprises 20 solder pins or wire connections. This allows other applications to be tried out and so can be linked to a suitable 'real' connector either at the rear of the case or on the control panel. It is advisable to use a 31 pin (female) type similar to the PIA port connector on the main board and use ribbon cable as the connection medium.

o. Finally, the **output connector**. This connector is only required if extra bus board memory is to be used and even then it is not strictly necessary, as the bus board can be connected with ordinary wires. Readers who prefer to use a connector should use the type illustrated in figure 19c and mount it on the copper side of the board. There are a few minor problems involved here, but fortunately they are quite easy to solve. The output connector cannot be mounted in the normal manner by inserting it and soldering the 64 pins. In view of the position of the two rows of 32 holes, the connector cannot be placed on the edge of the board itself. This is because both sides of the board have had to be covered in copper tracks to save space. The pins might just pass through to the other side of the board at a pinch, but it is wiser to choose **one** of the following solutions:

- Using a pair of 'snipe-nosed' pliers the pins can be bent slightly to gain an extra few millimetres.
- Take a connector of the type shown in figure 19b with pins that are at least 13 mm long (wire-wrap type) and again bend them so that they fit easily into the holes.
- Take a connector of the type shown in figure 19b with shorter pins and connect it to the board by means of 64 wires (hardly ideal!).
- If the distance between the board and the connector gets any greater (due to bus board memory being added via connector at the rear of the case, for instance) it is better to use ribbon cable instead of the 64 individual wires.

Figures 20b and 20c will help you make up your mind in this matter as will step four, which we are about to discuss right now.

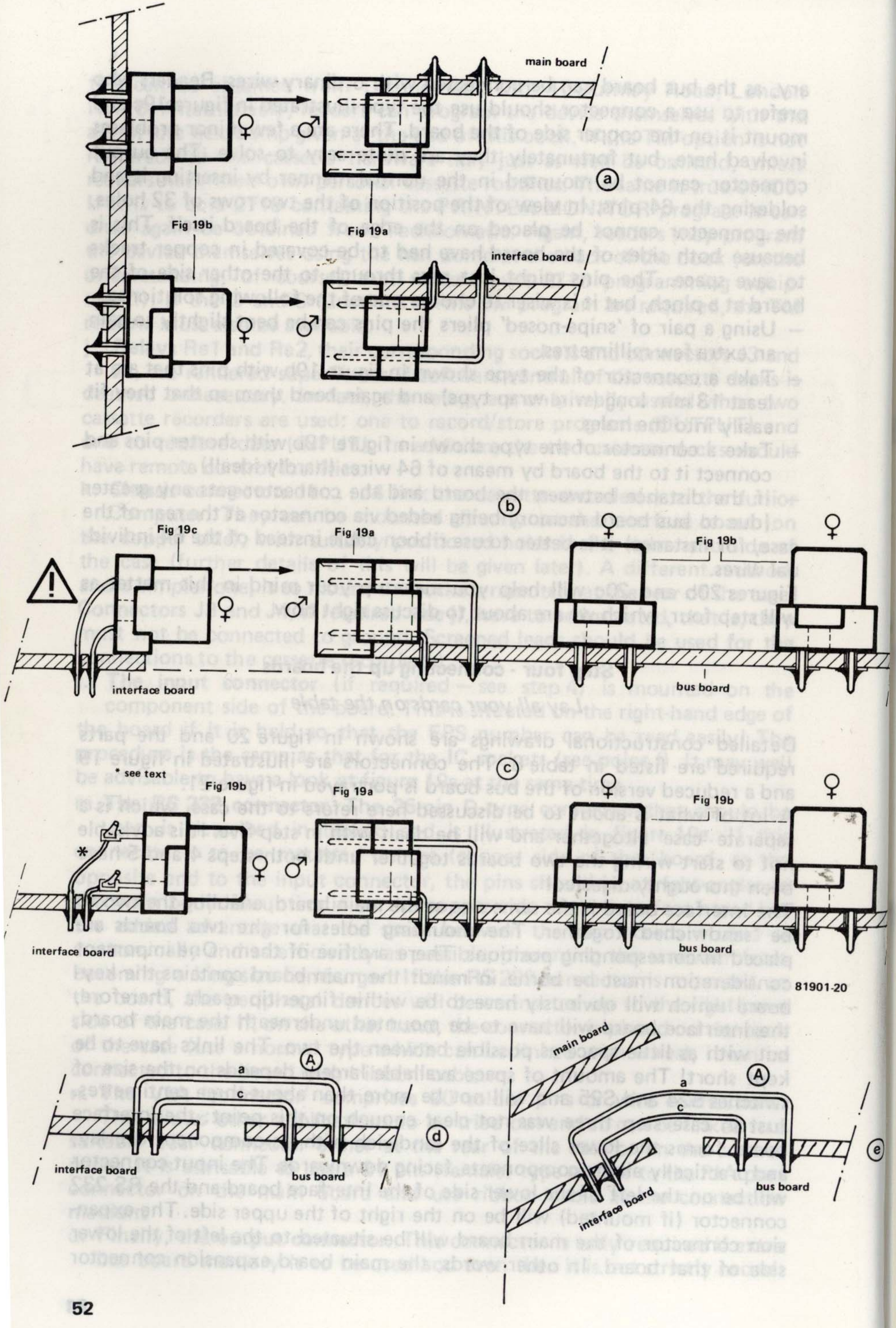
Step four - connecting up the boards

Lay all your cards on the table

Detailed constructional drawings are shown in figure 20 and the parts required are listed in table 8. The connectors are illustrated in figure 19 and a reduced version of the bus board is portrayed in figure 21.

A lot of what is about to be discussed here refers to the case, which is a separate 'case' altogether and will be dealt with in step five. It is advisable not to start wiring the two boards together until both steps 4 and 5 have been thoroughly digested!

The interface board is the same size as the main board, enabling the two to be 'sandwiched' together. The mounting holes for the two boards are placed in corresponding positions. There are five of them. One important consideration must be borne in mind: the main board contains the keyboard which will obviously have to be within finger-tip reach. Therefore, the interface board will have to be mounted underneath the main board, but with as little space as possible between the two. The links have to be kept short! The amount of space available largely depends on the size of switches S24 and S25 and will not be more than about three centimetres. Just in case step three was not clear enough on this point: the interface board forms the lower slice of the sandwich with its component overlay, and practically all the components, facing downwards. The input connector will be on the left of the lower side of the interface board and the RS 232 connector (if mounted) will be on the right of the upper side. The expansion connector of the main board will be situated to the left of the lower side of that board. In other words, the main board expansion connector



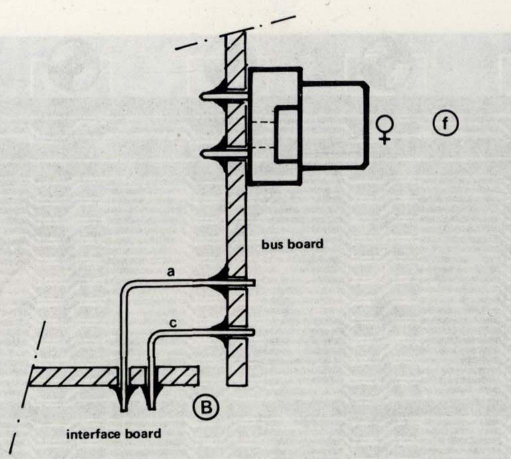


Figure 20. Detailed view of the electrical connections between the main and interface boards (20a) and the interface and bus boards (20b . . . 20f).

Table 8. Electrical connections for the entire unit.

Electrical connections for the entire unit

a. **between main board and interface card**
2 64-pin connectors, female (see figure 19b)
1 printed circuit board EPS 80024 (partly used)
1 31-pin male connector (see figure 19d)

b. **between interface card and bus board**
1 64-pin male connector placed at right angles (see figure 19a)
1 . . . 5 64-pin female connector(s) (see figure 19b)
1 printed circuit board EPS 80024

and the interface board input connector will be directly above each other on the left-hand side.

Next, the bus board. The output connector is placed on the copper side and on the long edge of the interface board, the edge furthest away from the operator and closest to the input connector, at the rear left. The bus board itself 'stretches' away from the operator with 1 . . . 5 memory cards mounted vertically on it.

Now that we have a good idea of the way in which the various boards should be placed, it is time to connect them all up. Here again, there are various possibilities, depending on the type of case selected to house the completed unit.

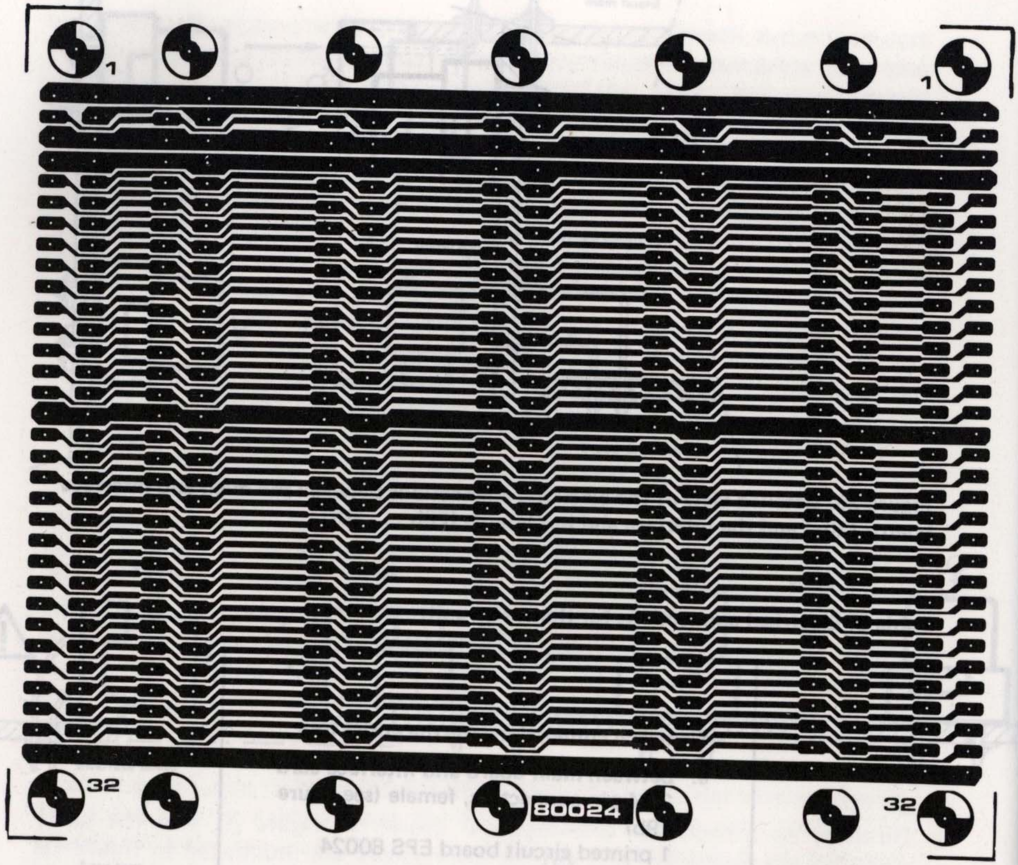


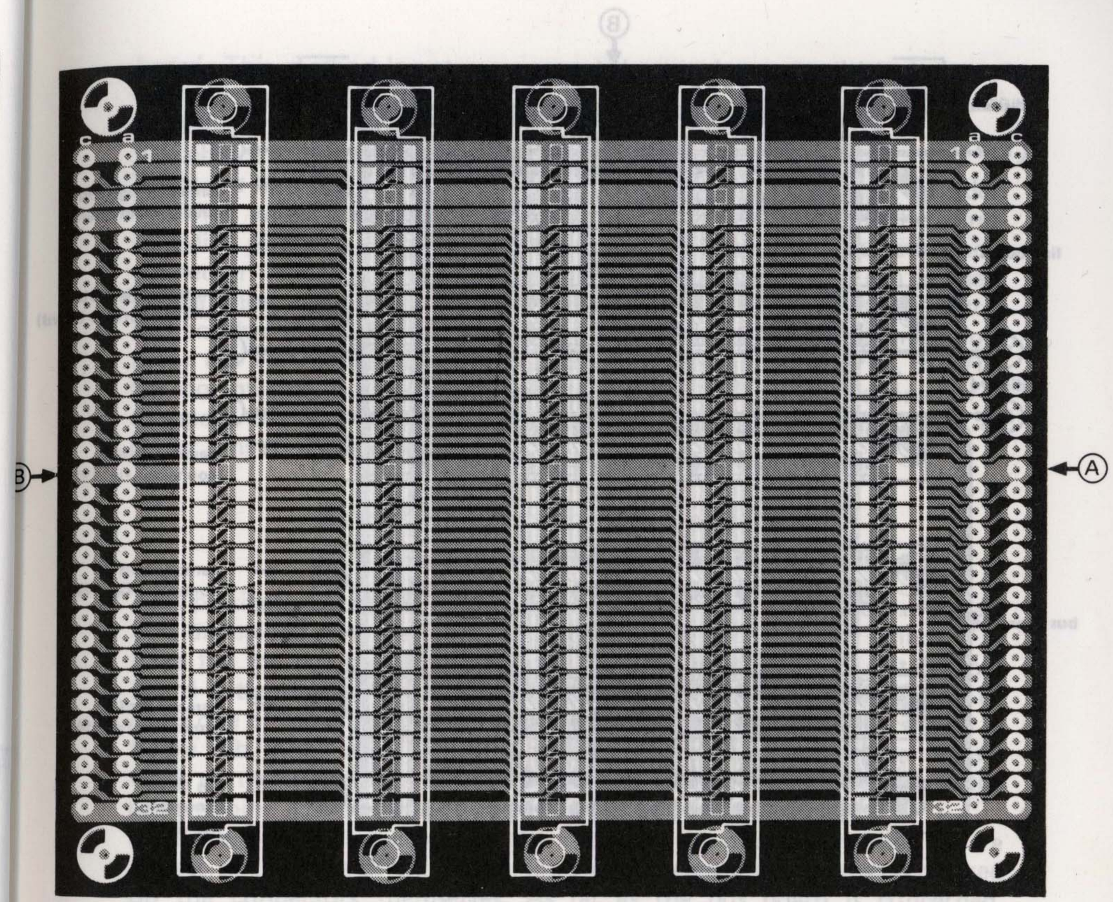
Figure 21. The printed circuit board and connector overlay for the bus board. The 'cutting' details are also given (21b).

Connecting the main and interface boards together

The 64 'stitches'

There are a number of preferred methods of actually making the connection between the two boards:

1. Use **straight** connectors (instead of right-angled ones) on both boards. The main and interface boards will then simply fit together (provided of course one connector is male and the other female!). The main point to watch here is the distance between the two boards, which is determined to a certain extent by the crystal and switches on the main board. The switches can be moved, as mentioned earlier, but the crystal cannot! However, there is no reason why the crystal cannot be re-sited on the other side of the board. Note that if this method is employed, the mounting holes for the connectors will not line up. This should not present a problem however, as the connection is only likely to be made once! There is ample 'strength' when all 64 pins are soldered.



2. The two boards can be joined together by using a pair of connectors as shown in figure 19b which are linked together by means of a short length of ribbon cable.
3. Use **part of** a bus board. The bus board has been mentioned on several occasions, but without fully describing it. We are referring to the SC/MP bus board (EPS number 80024) which was published in the January 1980 issue of Elektor and which is shown in figure 21. The idea is to use one end section containing two memory card connections to link the two boards. The remaining three connections can be used for the actual bus board if only three memory cards (or less) are to be added.

A storm of protest all over the UK! What? Cut up such a nice board? Well it's only an idea, but it works (see figures 20a and 21b). Note that the copper tracks on the bus board are not symmetrical. Pins 3 and 4 are linked to broad copper tracks, but if the board is turned around, the corresponding pins on the other side lead to much thinner tracks. Looking at the sandwich 'edge-on', as shown in figure 20a, pins 32 are closest to the operator and pins 1 are furthest away.

4. No connectors are used at all, in other words, neither are the expansion and input connectors. The various links are all made directly to each

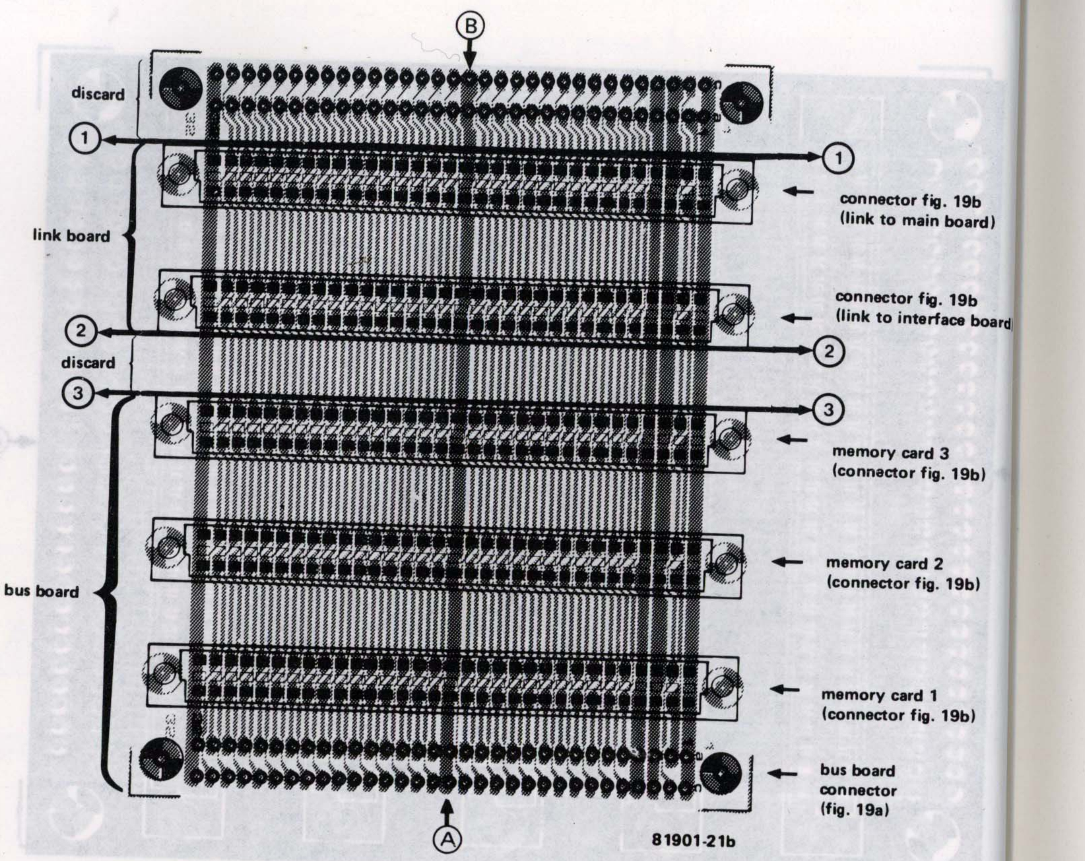

B
discard
1
1
link board
connector fig. 19b (link to main board)
connector fig. 19b (link to interface board
2
2
discard
3
3
memory card 3 (connector fig. 19b)
bus board
memory card 2 (connector fig. 19b)
memory card 1 (connector fig. 19b)
bus board connector (fig. 19a)
A
81901-21b

Figure 21b. Readers who do not require more than three extra memory cards can use part of the bus board (EPS 80024) to link the main and interface boards. This figure shows the purpose of the various sections. The area above the line ①-① can be discarded, as can the section between lines ②-② and ③-③. The area between lines ①-① and ②-② can be used as the link board and the remaining section can be used as the actual bus board.

board. This method is not recommended, as it demands a high degree of soldering expertise and, as most experts will agree, connectors are far more suitable!

The choice is up to you!

There are also the five connections to the PIA port connector to consider. The five wires connected to the main board are soldered to the corresponding pins of a 31 pin connector which is fitted into the existing port connector. Alternatively, of course, five individual solder pins can be used, or the wires may be soldered directly to the pins of the port connector. Yet another method is to forget about the connector altogether and to use it for the VIA connector at the rear of the case (unlike the PIA, the VIA has no internal 'housekeeping' jobs to perform). If the port connector is dropped, the five wires can be soldered directly to the main board, or solder pins could be employed.

Wiring the interface board to the bus board

Data transfer to higher addresses

This particular section is meant for 'bus trippers' only, but readers who are still undecided now have the opportunity to make up their minds. The links for the bus board are shown in figures 20b and 20c. A suitable bus board is the one described previously (EPS number 80024, see figure 21), using one of the connectors shown in figure 19a and five of those shown in figure 19b. When using the bus board, remember the asymmetrical copper track pattern. In the 'sandwich' position stipulated, pins 32 will be to the left and pins 1 to the right.

In all the 64 pin connectors on the sandwich the pin rows marked (a) are closest to the edge of the board. This also corresponds to what is marked on the connectors themselves. As far as the bus board is concerned, however, this is the exact opposite: the (c) rows are closest to the edge. For this reason a particular copper track will be marked (a) on the input connector and (c) on the output connector, and vice versa (see figure 1).

Note: All the connections marked (a) and (b) given in figure 19 correspond to those on the connector, not to those marked on the bus board!

If the interface board and the bus board are linked by means of 64 wires, without any connectors, the wires connected to the (a) pins will be just as long as those connected to the (c) pins (see figure 20d). If proper connectors are used instead, there are two possibilities (figures 20b and 20c) with regard to the position of the output connector discussed before (either on the board or to one side of it). Note the difference in height between the drawings in figures 20b and 20c: they are at different levels in figure 20b, whereas in figure 20c they are at the same level.

If a 'lectern' type case is selected to house the completed Junior Computer, the main board is likely to be at an angle of some 45°, so therefore the interface board will have to be at an angle as well (unless they are mounted as shown in figure 22). In view of all the connections, the bus board should preferably be mounted horizontally (or vertically, as explained below). If the bus board is mounted horizontally, the only way to connect it is as shown in figure 20e. Connectors can not be used, unless a length of ribbon cable is also used.

The bus board can also be mounted vertically. This situation is shown in figure 20f. Pins 1 and 32 maintain the same position, but the bus board is linked up on the opposite side to that previously. The case should then be a long flat box, as a bus board is shorter than a memory card. Thus, the operator has plenty of choice as far as the case is concerned, which is just as well as the number of different types available nowadays is almost unbelievable!

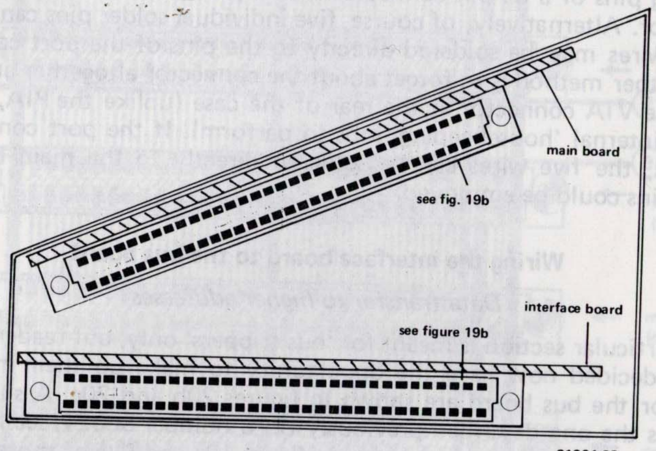


Figure 22. If a 'lectern' type case is used to house the extended Junior Computer, a separate 'connection' board may come in handy to link up the main and interface boards.

Step five - housing the fully-fledged Junior Computer

The final assembly

Modern houses nowadays are often built with 'everything under one roof', including the garage. This is an idea which could well be applied to the Junior Computer. At the moment the power supply is probably housed in a separate case because it takes up so much space. Some people, however, might prefer to include everything in a single case. This certainly looks a lot neater, although the case will then, of course, have to be somewhat larger and stronger, but at least there will be no need to worry about mixing up the various connections.

The number of 'houses' also depends on which peripheral devices are to be used. As it is, there will have to be a case for the cassette recorder(s), one for the ASCII keyboard and video terminal (Elekterminal) and, of course, the television set. Putting the whole 'village' into a single case is easier said than done and besides, think of the weight and the cost of the case! The various items may just as well be housed in separate cases, meaning that the basic version of the Junior Computer does not really have to 'move'!

However, if the main board and the various extension boards are to be installed in the same case, there are two main possibilities: a 'lectern' model and an 'amplifier' model. As its name suggests, the **amplifier** model (see figure 23b) looks similar to the type of case used for the old type of audio (stage) amplifiers, equipped with obsolete valves etc. Believe it or not, this type of case is still available and is ideal for the 'all-in-one' solution. The board sandwich is mounted under the slanting control panel, with the bus board and associated memory cards situated at the rear left and the power supply including the two transformers to the rear right.

The **lectern** model (see figure 23a) was probably the type used for the original version of the Junior Computer, albeit not exactly the same as that shown in figure 23a. This type of case is lot more compact than the amplifier version and forms the centre of the computer 'village'. Here are a few considerations to help readers make their final choice:

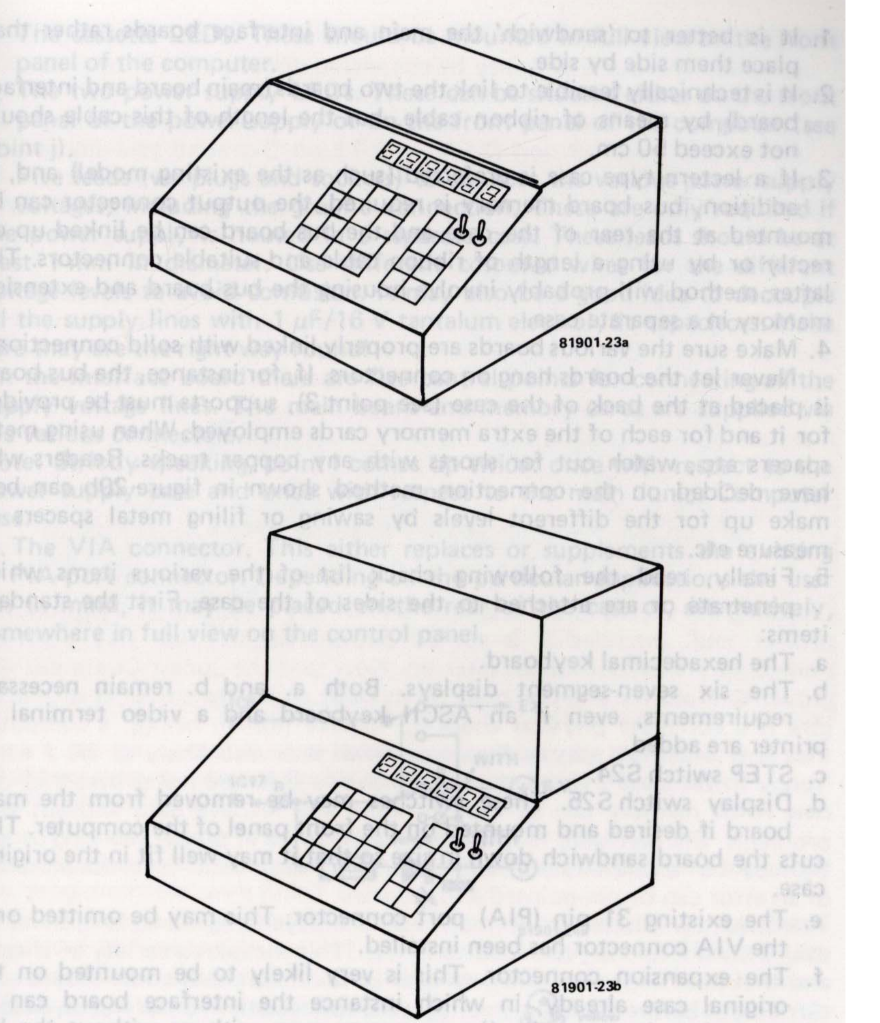


Figure 23. There are two main types of cases in which the 'mature' Junior Computer (with all the extensions) could be housed. It all depends on whether the operator wishes to have everything inside one large case, or spread over several cases. The two types may also be combined in a variety of ways.

1. It is better to 'sandwich' the main and interface boards rather than place them side by side.
2. It is technically feasible to link the two boards (main board and interface board) by means of ribbon cable, but the length of this cable should not exceed 50 cm.
3. If a lectern type case is preferred (such as the existing model) and, in addition, bus board memory is required, the output connector can be mounted at the rear of the case and the bus board can be linked up directly or by using a length of ribbon cable and suitable connectors. The latter method will probably involve housing the bus board and extension memory in a separate case.
4. Make sure the various boards are properly linked with solid connections. Never let the boards hang on connectors. If, for instance, the bus board is placed at the back of the case (see point 3), supports must be provided for it and for each of the extra memory cards employed. When using metal spacers etc., watch out for shorts with any copper tracks. Readers who have decided on the connection method shown in figure 20b can best make up for the different levels by sawing or filing metal spacers to measure etc.
5. Finally, read the following check list of the various items which penetrate or are attached to the sides of the case. First the standard items:

a. The hexadecimal keyboard.

b. The six seven-segment displays. Both a. and b. remain necessary requirements, even if an ASCII keyboard and a video terminal or printer are added.

c. STEP switch S24.

d. Display switch S25. These switches may be removed from the main board if desired and mounted on the front panel of the computer. This cuts the board sandwich down in size so that it may well fit in the original case.

e. The existing 31 pin (PIA) port connector. This may be omitted once the VIA connector has been installed.

f. The expansion connector. This is very likely to be mounted on the original case already, in which instance the interface board can be connected to it externally (in a separate case, with or without the bus board memory). Unfortunately, this means deciding against the sandwich principle.

g. The mains cable, main switch and fuse holder have to be fitted to the power supply or main case. Do not forget the 2A fuse!

h. The RS 232 connector. This can be mounted either directly on to the side of the interface board or at the back of the case by means of ribbon cable. Make sure the links are in the correct place. The orientation of the D-type connector is such that it has to be mounted on the copper side of the board. If the Elekterminal is used, the links between the pins (see figure 1) may be omitted.

i. The chassis connectors for the cassette recorder(s) are best mounted at the rear of the case. As an alternative to J1 and J2 a DIN connector can be used. Connectors J3 and J4 are only required if the recorder(s) is (are) to be controlled via software.

j. The cassette LEDs. These should be mounted in full view on the front panel of the computer.

k. The two power supply LEDs. These can be situated either on the front panel of the power supply or on the front panel of the computer (see point j).

l. Five leads (via plugs and sockets) to connect the various power supply voltages, including the ground connection. These are only required if the power supply is housed in a separate case. These leads should be at least 1 mm in diameter. Use different coloured wires for the different voltage levels to avoid confusion. It may also be a good idea to decouple all the supply lines with 1 μF/16 V tantalum electrolytic capacitors. Make sure they are the right way round!

On the interface board there are five central points for connecting all the supply voltage lines. The main board and memory cards are supplied via the various connectors.

Note: Strictly speaking, point l comes up twice: once with respect to the power supply case and once with respect to the main Junior Computer case.

m. The VIA connector. This either replaces or supplements the existing PIA port connector. Depending on the particular applications the user has in mind, it may be placed on the rear of the case or, alternatively, somewhere in full view on the control panel.

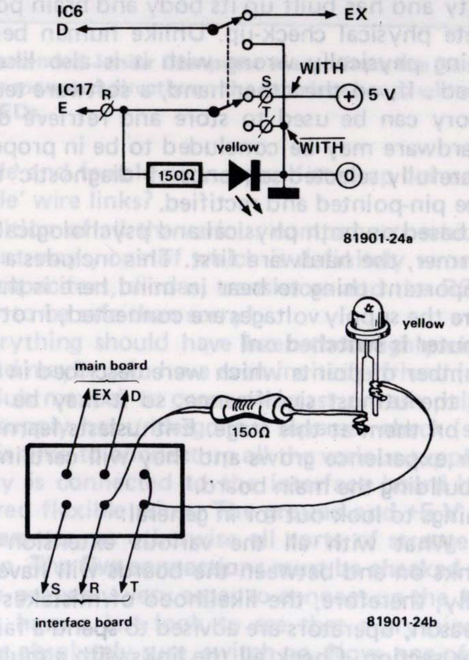


Figure 24. A WITH/$\overline{\text{WITH}}$ switch can be incorporated if desired. This allows the user to select bus board memory if and when required. This offers an alternative to the 'hard' wire links.

n. The output connector. This is only required if the bus board option is selected. The bus board may be connected directly to it or by means of ribbon cable and a suitable male connector.

o. The WITH/$\overline{\text{WITH}}$ switch is none other than a double-pole switch that selects the links R-S and D-EX or R-T and D-ground (see figure 24). This needs to be added if bus board memory is to be used and so it is as well to include it now even if you are not ready to incorporate extra memory just yet. In the WITH position a yellow LED will light, so this should also be in a clearly visible position.

We have now reached the end of the constructional details, so readers may now heat up their soldering irons (which are probably white hot anyway) and set to work. Obviously, no description can be foolproof and we accept the fact that the various options may complicate matters considerably. On the other hand, most operators will appreciate such constructional freedom and it will be interesting to see how many different versions are created!

Step six - Check for possible errors

The end of the road

Checking for errors is an essential part of any electronics constructional project and, provided it is carried out thoroughly, will save a lot of unnecessary disappointment later on. Now that the Junior Computer has grown to maturity and has built up its body and brain power, it is time to give it a complete physical check-up. Unlike human beings, a computer that has something physically wrong with it is also likely to be a little 'funny in the head'. If, on the other hand, a software test proves that the new RAM memory can be used to store and retrieve data without any problems, the hardware may be concluded to be in proper working order. By means of a carefully selected sequence of 'diagnostic' tests the presence of an error can be pin-pointed and rectified.

The check-up is based on both physical and psychological factors, so let us deal with the former, the hardware first. This includes a variety of checks and the most important thing to bear in mind here is the fact they are all carried out **before** the supply voltages are connected, in other words, **before the Junior Computer is switched on!**

Here again, a number of points which were described in Book 1 (pages 31 and 32) are of the utmost significance, so it may be advantageous for readers to recap on them at this stage. Enthusiasts learn to watch out for mistakes as their experience grows and they will certainly have learned a great deal from building the main board.

To start with, things to look out for in general:

1. **Connections.** What with all the various extension possibilities, the number of links on and between the boards will have increased alarmingly. Statistically, therefore, the likelihood of mistakes is proportionally high. For this reason, operators are advised to spend a fair amount of time on this particular section. Check all the links with a multimeter set on the resistance range, or with the continuity tester described in the July/August 1981 issue of Elektor. Are all the IC pins inserted correctly? Make sure that none of the links are round the wrong way or connected to the wrong pins. Are the male and female connectors fitted up correctly? What about the 'programmable' wire links?

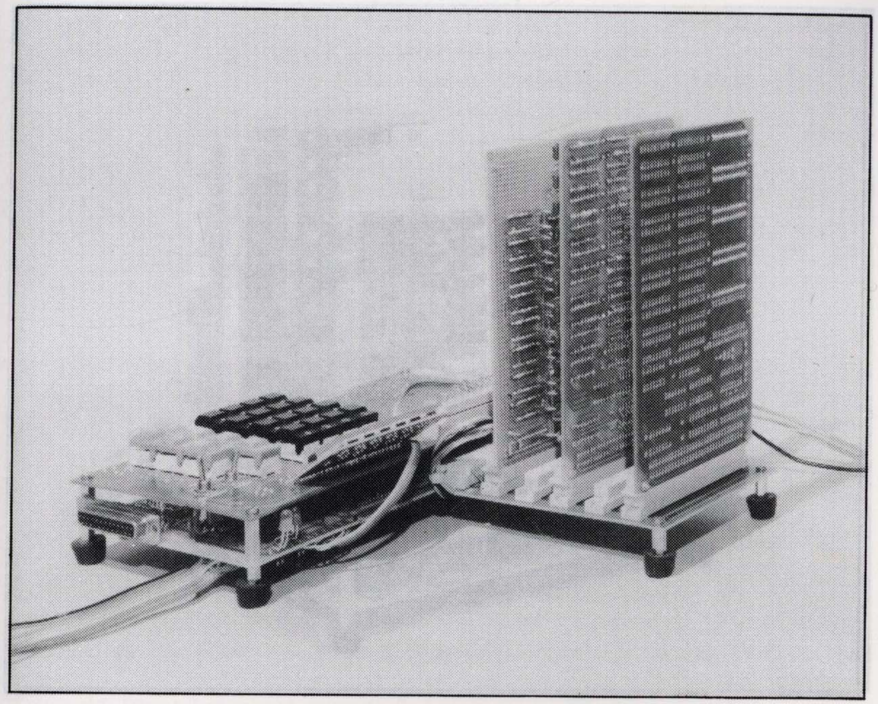


Photo 1. The fully extended Junior Computer as seen from the North-East. Here the RS 232 connector is mounted directly on the interface board and so are the INPUT/OUTPUT LEDs.

2. Check the position of all the various components which can be placed in two different ways, one of which is definitely wrong. This includes ICs, electrolytic capacitors, diodes, transistors and the RS 232 connector. It is always better to be safe than sorry!

By this time everything should have been thoroughly checked out from top to bottom and maybe you have even invited a friend (a member of the local computer club maybe) to come and cast a fresh pair of eyes over it. Once the power supply has undergone a separate check (see the constructional details), it is time to connect up all the various supply voltages.

The power supply is connected to the interface board by means of five differently coloured flexible wires. **The ground and +5 V leads must be at least 1 mm in diameter,** as otherwise all sorts of strange and even nasty things may happen. **The five connections must be checked thoroughly!**

Although you are probably very eager to connect up the Junior Computer and switch it on, have a last look to see that everything is really 100% correct. If you are absolutely sure, switch on. Now, one of two things may happen: your impatience gets away with you and you depress the RST (reset) key. If all the displays light, the computer will be operating inside the original section of the extended Junior Computer. Either that, or you

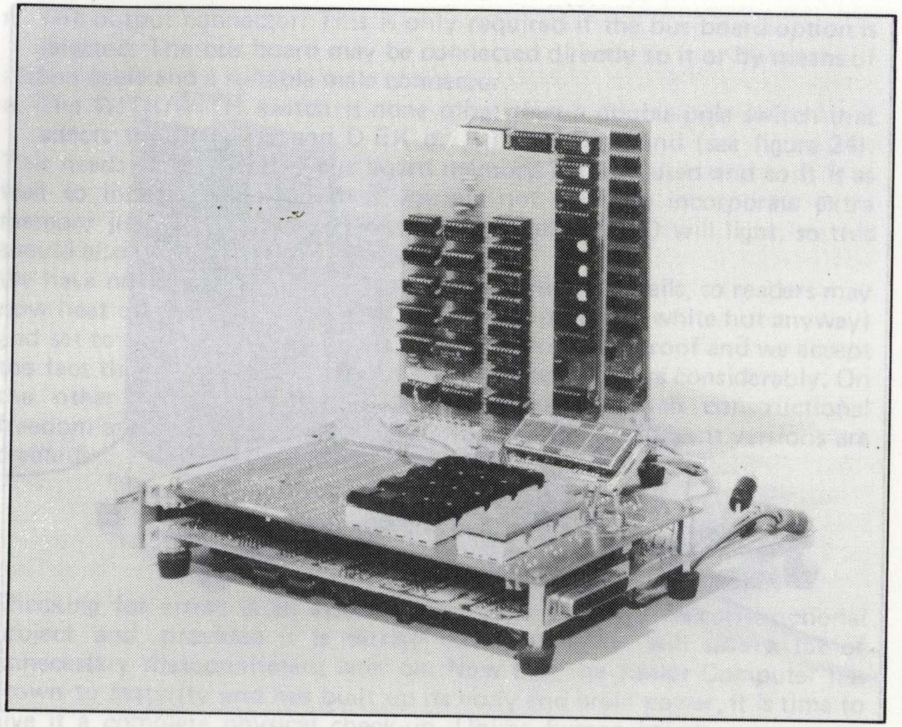


Photo 2. The same as photo 1, but now as seen from the South-East. Note that the copper side of the interface board is facing upwards. Female chassis connectors are used to link the computer to one or two cassette recorders. The nearest memory card is a prototype of the forthcoming 16k dynamic RAM card.

patiently measure the various voltages connected to the IC pins using the information given in figure 18. **Test the voltages are correct at the IC pins — not at the solder points on the copper side of the board!** This is the quickest way to detect a faulty IC contact. Make sure that the probes do not cause a short with neighbouring pins, as that would really make matters worse.

Now the display should light when the RST key is depressed. If this fails to happen, and you are sure that the main board is perfectly all right, the following questions have to be answered: Is the main board being supplied with enough power? What about the wire link at point D of IC6? This should be linked to EX if extra memory is connected. In the latter case, check that there is an EPROM connected to page FF and that the RESET vector (addresses FFFC and FFFD) is specified correctly (FFFC = 1D, FFFD = 1C). Is the wire link R-S installed? Does the main board work when both wire links D-ground and R-T are connected temporarily? What about removing the interface board and using temporary power supply connections as before? The interface board hardware around the address decoding system and the data bus control require particular attention, including the PROM, IC17. By the way, is this programmed correctly?

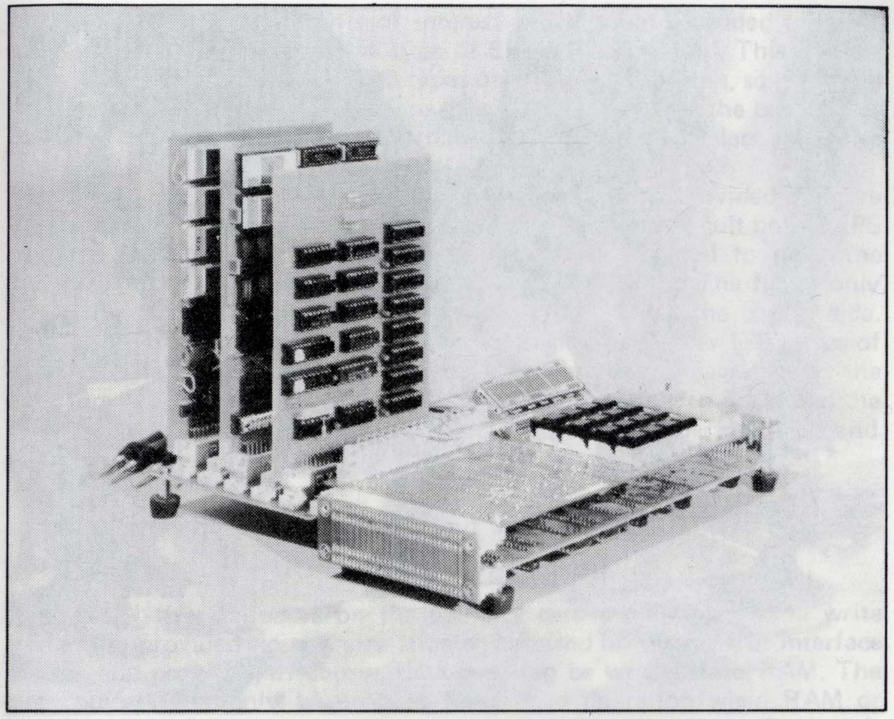


Photo 3. The same as photo 1 only as seen from the South-West. The expansion connector of the main board and the input connector of the interface board are linked by means of part of a bus board. Also shown is the connection to the port connector of the main board.

As soon as the basic monitor routine runs like clockwork, it becomes a good tool to test the rest of the equipment, such as the new RAM, or in any case IC2 and IC3 and, if 8114 type devices are used, IC4 and/or IC5. This is how it works. A few addresses inside the corresponding RAM address range are selected. One address is keyed in with the aid of the AD key. Data then appears on the two right-hand displays. By depressing the DA key new data, different to that displayed, can be entered. If the new data also appears in the display, the operator can be sure that the RAM memory location concerned is being written to and read from correctly (see chapter 7 in Book 2). Repeat this procedure for several other addresses in the same range (using the + key for instance) and in all the other RAM address areas.

EPROMs have to be read properly. They are only written into (programmed) once. If IC4 and IC5 contain the TM and PM routines, respectively, run the monitor program to see if these are read properly. The hex dumps for both routines are given at the end of this book.

Left to check are the VIA, the cassette and the RS 232 interfaces. Before the cassette interface hardware can be fully operational, the PLL has to be calibrated first with the aid of the trimmer potentiometer P1. More about

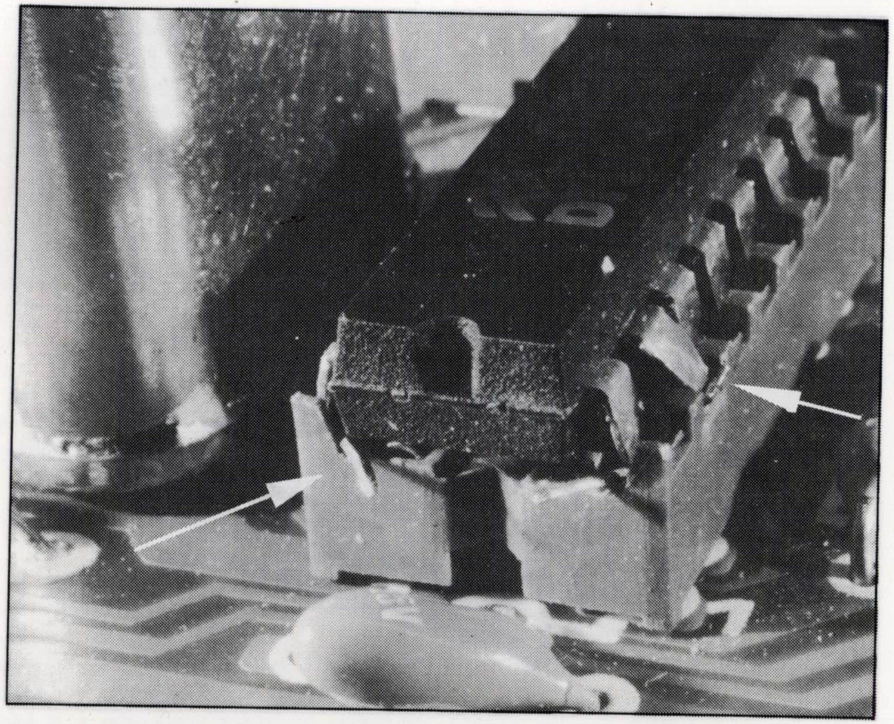


Photo 4. This photo shows what can go wrong when installing ICs!!

this in chapter 11. The VIA is dealt with in Book 4 and the RS 232 interface cannot be checked until all the corresponding peripheral devices are built and connected up. This does not occur until chapter 12.

The fully extended Junior Computer is now complete and tested. A number of operations cannot be fully checked until chapters 11 and/or 12 have been read. As far as the bus board memory is concerned, this depends on the RAM/EPROM card, which has to be especially prepared for use. This is in fact the third and final subject for discussion in this particular chapter.

The RAM/EPROM card

How to reach 'outside' addresses by bus

The RAM/EPROM card, like the 16k dynamic RAM counterpart (see Book 4), allows up to 56k of extra memory to be added to the extended Junior Computer. This amounts to pages 2Ø . . . FF. In addition, one memory card can be linked directly to the main board, in other words, this is one memory area which does not require a bus connection. It makes do with the expansion connector instead. This option is described in Appendix 1.

The RAM/EPROM card was first described in the September 1980 issue of Elektor, so there is no need to repeat the whole story here. There are, however, a number of points of interest which could be added to it. Be very careful when choosing the type of EPROM to be used. This may be 1 . . . 4 2708 types, 1 . . . 4 2716 types or 1 . . . 4 2732 types, so please do not start mixing them up! Of these, the 2716 is probably the best choice as it has more memory capacity than the 2708 and is far less expensive than the 2732.

The circuit diagram of the entire RAM/EPROM card is provided in figure 25. Figure 26 contains a reduced version of the printed circuit board, EPS number 80120. In actual fact, this board was designed to meet the eurocard format requirements and is 100 mm x 160 mm. The figure only shows the component overlay and the track pattern for the copper side. All the other features are shown full-scale in the September 1980 issue of Elektor. The pin assignments and the internal circuit diagrams for the decoders, IC5 . . . IC7, are shown in figure 27a. Figure 27b illustrates the truth tables of the main decoder IC5 and the secondary decoders IC6 and IC7.

Additional information about the RAM/EPROM card

Putting a few things straight

The data buffers situated on the memory card are enabled for a write operation, provided no memory is being accessed on the main or interface boards and provided, of course, data needs to be written into RAM. The data buffers will only be enabled for a read operation when RAM or EPROM (decoded on the card) is being read. Thus provided the card is used properly, the risky situation illustrated in figure 6 should not occur.

A bird's eye view of the operation of the main address decoder, IC5, is given in table 9. See figure 27b as well. The address range is split up into sixteen blocks of 4k. In the September 1980 article we referred to these blocks as 'pages'. This could lead to a little bit of confusion since, as far as the Junior Computer is concerned, a page consists of 256 (= FF) bytes. Further confusion is caused by the fact that address lines A12 . . . A15 are not connected to IC5 in a very logical manner. In addition, table 9 shows that in a particular 4k memory area only the most significant nibble of the address (Øxxx . . . Fxxx) is determined. The other three (x) can each be any hexadecimal number between Ø and F.

Secondary address decoding for the RAM section takes place via IC6 which is controlled by three lower address lines (A10 . . . A12) and by gate N2. Up to 8k of RAM can be addressed. That is why IC6 has eight outputs (see figure 27b) and N2 has two inputs which receive 'select' data from the main decoder, IC5 (see connections X and Y). If only 4k of RAM is used, one connection to N2 is sufficient.

Secondary address decoding for the EPROM section takes place by way of IC7 (see figure 27b). Depending on the type of EPROM used, it will be controlled by two lower address lines and gate N1, which is connected to IC5 by means of links V and W.

When selecting particular memory areas on the card, the possibilities are virtually endless, as can be seen from the 'memory map' for RAM and EPROM. Fortunately, the memory map can be organised quite well. If up to 56k of extra memory is connected to the computer, this being the

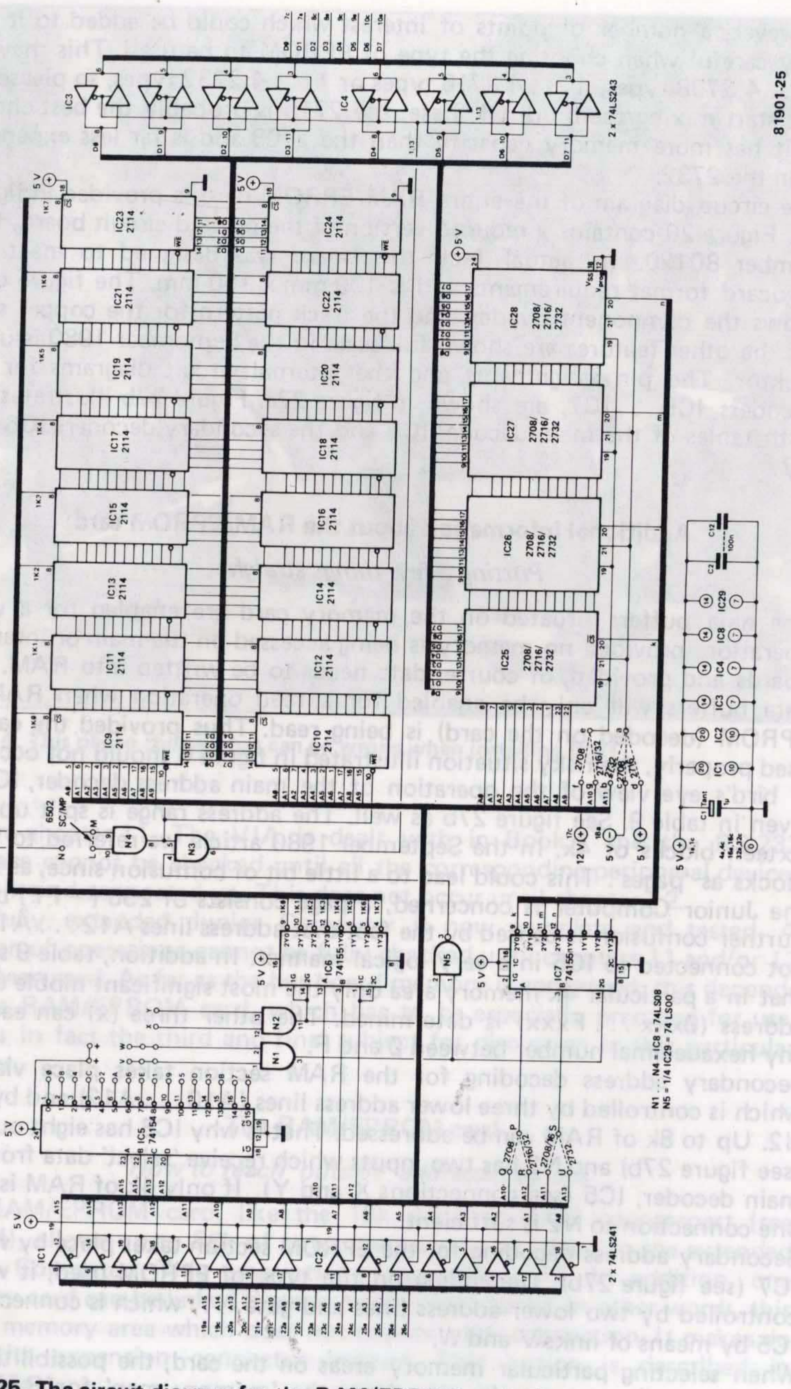


Figure 25. The circuit diagram for the RAM/EPROM card. There are different methods of connecting the 2716 links indicated and these are mentioned in the appropriate text.

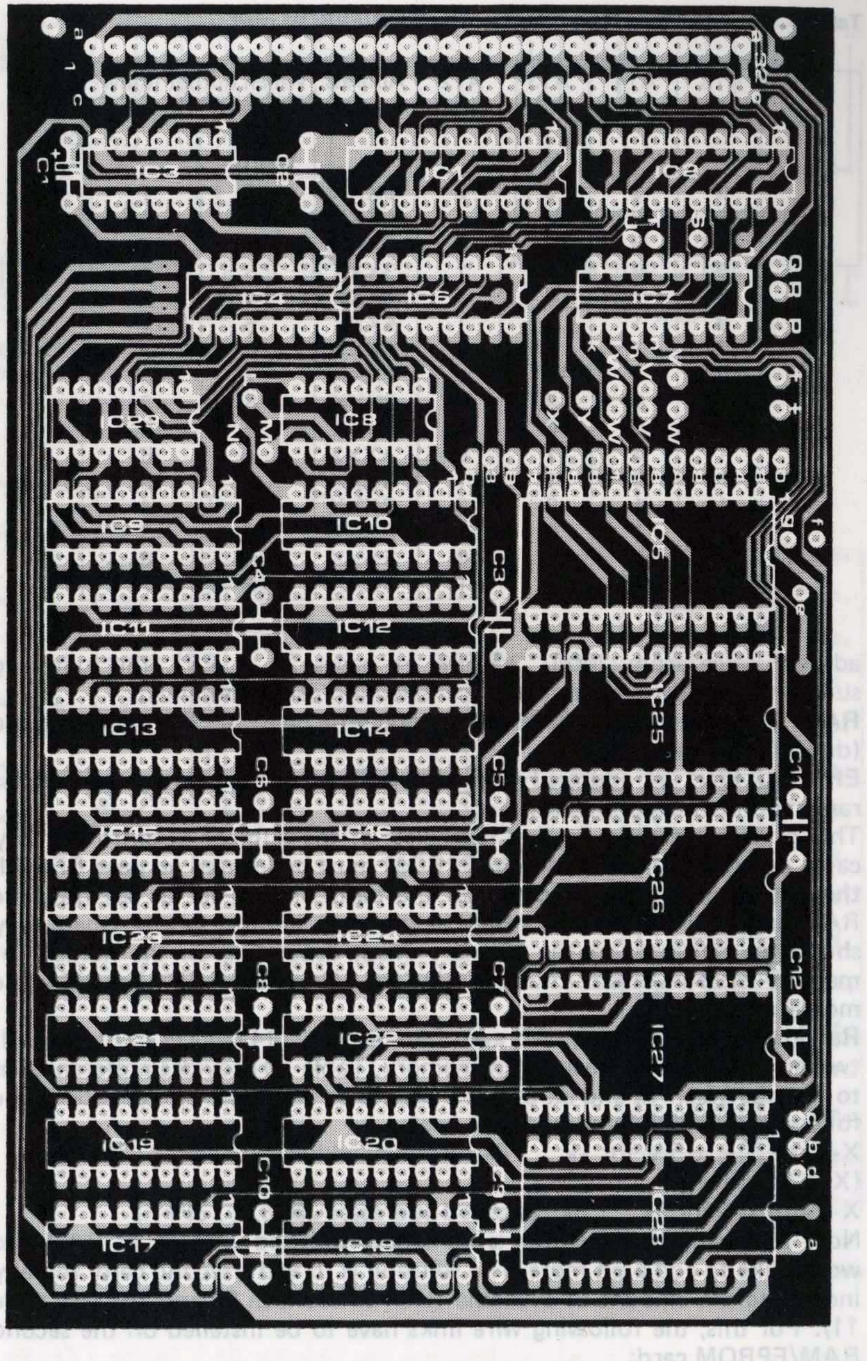


Figure 26. The printed circuit board for the RAM/EPROM card showing the component overlay and the track pattern of the copper side. Again, this is shown reduced in size.

Table 9. The main decoding system on the RAM/EPROM card.

A15 (A)	A14 (B)	A13 (C)	A12 (D)	Address (hex)	Which output is low? connec-tion on board	 IC5 pin
Ø	Ø	Ø	Ø	ØXXX	Ø	1
Ø	Ø	Ø	1	1XXX	1	9
Ø	Ø	1	Ø	2XXX	2	5
Ø	Ø	1	1	3XXX	3	14
Ø	1	Ø	Ø	4XXX	4	3
Ø	1	Ø	1	5XXX	5	11
Ø	1	1	Ø	6XXX	6	7
Ø	1	1	1	7XXX	7	16
1	Ø	Ø	Ø	8XXX	8	2
1	Ø	Ø	1	9XXX	9	10
1	Ø	1	Ø	AXXX	A	6
1	Ø	1	1	BXXX	B	15
1	1	Ø	Ø	CXXX	C	4
1	1	Ø	1	DXXX	D	13
1	1	1	Ø	EXXX	E	8
1	1	1	1	FXXX	F	17

address range from 2ØØØ . . . FFFF (= pages 2Ø . . . FF), the following survey shows how a clear and logical selection can be obtained:

RAM: Start at address 2ØØØ and work your way up the address range (down the memory card);

EPROM: Start at address FFFF and work your way down the address range (up the memory card).

This principle can be applied regardless of the number of extra memory cards employed. For one thing, **EPROM must be located on page FF with the corresponding vector data.** On the basis of this, let us see how the RAM and EPROM sections should be decoded, in other words, where they should be situated on the memory map, depending on the number of memory cards involved – not counting the 16k dynamic RAM card for the moment.

Random Access Memory. An even number of 2114 devices must be used: two are required for each 1k of memory. Table 10 provides preferences as to how between 1k and 8k of RAM should be addressed and decoded. The following wire links are important here:

X-2 and Y – + 5 V for 1k . . . 4k (decoded addresses 2ØØØ . . . 2FFF)
(X-Y-2 is another possibility);
X-2 and Y-3 for 1k . . . 8k (decoded addresses 2ØØØ . . . 3FFF)

Now supposing the operator is no longer satisfied with 8k of RAM and would like to add a bit more. The solution is to add a second card, thereby increasing the amount of available RAM to between 9k and 16k (see table 11). For this, the following wire links have to be installed on the second RAM/EPROM card:

X-4 and Y+5 V for an **additional** 1k . . . 4k (decoded addresses 4ØØØ . . . 4FFF) – again, X-Y-4 is also possible;
X-4 and Y-5 for an **additional** 1k . . . 8k (decoded addresses 4ØØØ . . . 5FFF).

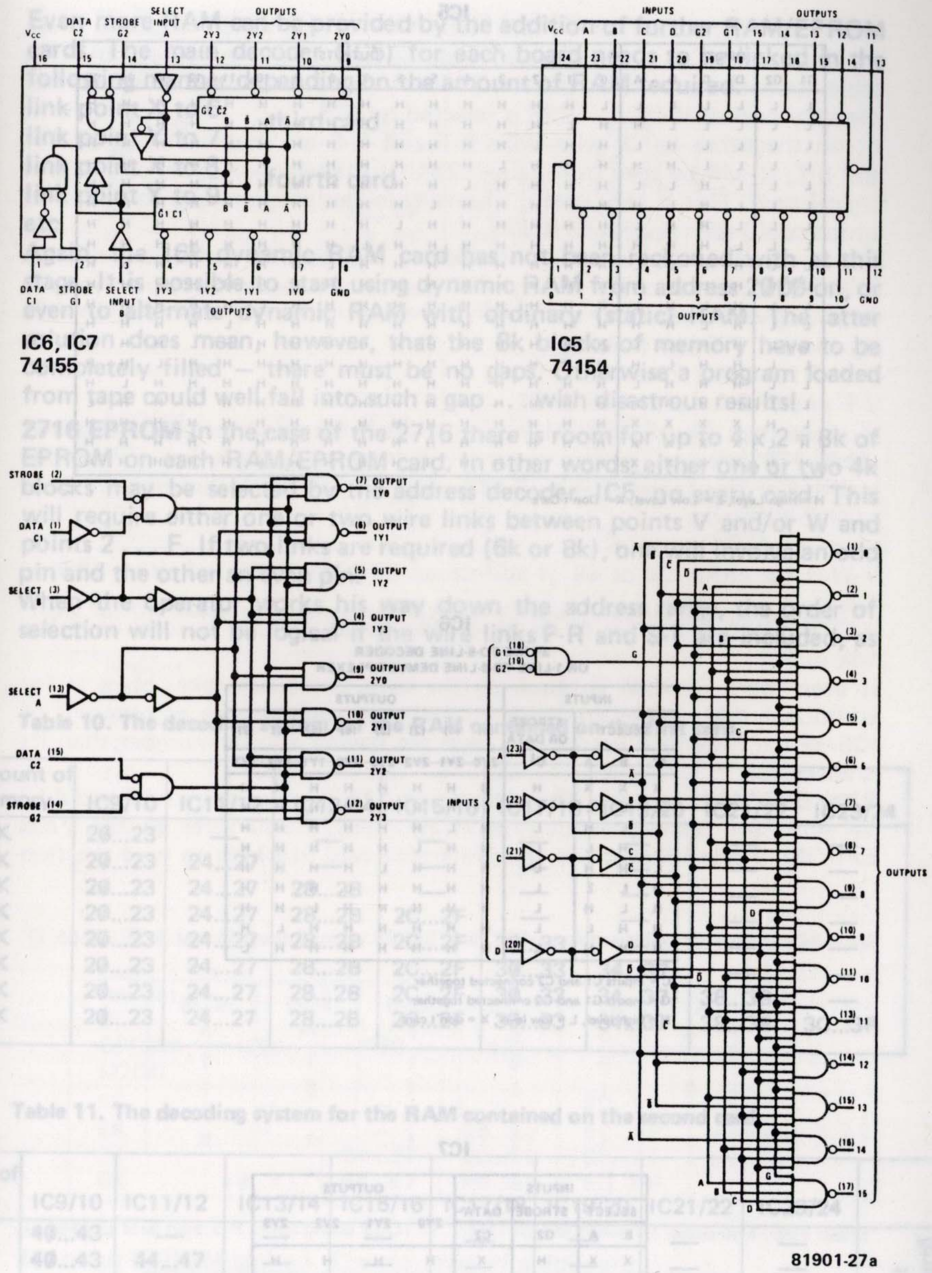


Figure 27. The pin assignments and internal circuitry of the decoder ICs contained on the RAM/EPROM card (27a). The truth tables referring to the main address decoder (IC5) and the secondary address decoders (IC6 and IC7) are also given (27b).

IC5

INPUTS						OUTPUTS															
G1	G2	D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	H	H	L	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

H = High Level, L = Low Level, X = Don't Care

IC6

3-LINE-TO-8-LINE DECODER OR 1-LINE-TO-8-LINE DEMULTIPLEXER

INPUTS				OUTPUTS							
SELECT			STROBE OR DATA	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
C[†]	B	A	G[‡]	2Y0	2Y1	2Y2	2Y3	1Y0	1Y1	1Y2	1Y3
X	X	X	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H
L	H	L	L	H	H	L	H	H	H	H	H
L	H	H	L	H	H	H	L	H	H	H	H
H	L	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	L	H	H
H	H	L	L	H	H	H	H	H	H	L	H
H	H	H	L	H	H	H	H	H	H	H	L

[†] C = inputs C1 and C2 connected together

[‡] G = inputs G1 and G2 connected together

H = high level, L = low level, X = don't care

IC7

INPUTS				OUTPUTS			
SELECT		STROBE	DATA	2Y0	2Y1	2Y2	2Y3
B	A	G2	C2				
X	X	H	X	H	H	H	H
L	L	L	L	L	H	H	H
L	H	L	L	H	L	H	H
H	L	L	L	H	H	L	H
H	H	L	L	H	H	H	L
X	X	X	H	H	H	H	H

81901-27b

Even more RAM can be provided by the addition of further RAM/EPROM cards. The main decoder (IC5) for each board needs to be linked in the following manner depending on the amount of RAM required:

link point X to 6 link point Y to 7	third card
link point X to 8 link point Y to 9	fourth card

etc.

Again, the 16k dynamic RAM card has not been reckoned with at this stage. It is possible to start using dynamic RAM from address 2000 on, or even to alternate dynamic RAM with ordinary (static) RAM. The latter solution does mean, however, that the 8k blocks of memory have to be completely filled – there must be no gaps. Otherwise a program loaded from tape could well fall into such a gap . . . with disastrous results!

2716 EPROM In the case of the 2716 there is room for up to 4 x 2 = 8k of EPROM on each RAM/EPROM card. In other words, either one or two 4k blocks may be selected by the address decoder, IC5, on every card. This will require either one or two wire links between points V and/or W and points 2 . . . F. If two links are required (6k or 8k), one will involve an odd pin and the other an even pin.

When the operator works his way down the address range, the order of selection will not be logical if the wire links P-R and S-T are included, as

Table 10. The decoding system for the RAM contained on the first card.

amount of memory	IC9/10	IC11/12	IC13/14	IC15/16	IC17/18	IC19/20	IC21/22	IC23/24
1K	2Ø...23	—	—	—	—	—	—	—
2K	2Ø...23	24...27	—	—	—	—	—	—
3K	2Ø...23	24...27	28...2B	—	—	—	—	—
4K	2Ø...23	24...27	28...2B	2C...2F	—	—	—	—
5K	2Ø...23	24...27	28...2B	2C...2F	3Ø...33	—	—	—
6K	2Ø...23	24...27	28...2B	2C...2F	3Ø...33	34...37	—	—
7K	2Ø...23	24...27	28...2B	2C...2F	3Ø...33	34...37	38...3B	—
8K	2Ø...23	24...27	28...2B	2C...2F	3Ø...33	34...37	38...3B	3C...3F

Table 11. The decoding system for the RAM contained on the second card.

amount of memory	IC9/10	IC11/12	IC13/14	IC15/16	IC17/18	IC19/20	IC21/22	IC23/24	
9K	4Ø...43	—	—	—	—	—	—	—	including 8K of Table 10 (first RAM/EPROM-card)
10K	4Ø...43	44...47	—	—	—	—	—	—	
11K	4Ø...43	44...47	48...4B	—	—	—	—	—	
12K	4Ø...43	44...47	48...4B	4C...4F	—	—	—	—	
13K	4Ø...43	44...47	48...4B	4C...4F	5Ø...53	—	—	—	
14K	4Ø...43	44...47	48...4B	4C...4F	5Ø...53	54...57	—	—	
15K	4Ø...43	44...47	48...4B	4C...4F	5Ø...53	54...57	58...5B	—	
16K	4Ø...43	44...47	48...4B	4C...4F	5Ø...53	54...57	58...5B	5C...5F	

shown in figure 25. This is illustrated in table 12a, which indicates when the particular IC is enabled (select input is logic zero, see figure 27b). This table applies when the output of N1 is logic zero, that is, when one of the memory devices IC25 . . . IC28 is selected by IC5.

If, on the other hand, wire links P-T and R-S are made instead of P-R and S-T, it will be seen from table 12b that the selection order certainly is logical now. This is very useful whenever longer programs are to be stored which could span several EPROMs.

The preferable way in which to address and decode the first (or only) EPROM section starting at page FF is shown in table 13. It involves the following wire links:

e-g and a-d; P-T and R-S,

V-W-F for either 2k or 4k;

V-F and W-E for either 6k or 8k.

If more than 8k of EPROM is required and the operator decides against using 2732 devices, a second card will have to be included on the bus board. For this table 14 should be examined and the following wire links have to be made:

e-g and a-d; P-T and R-S;

V-W-D for either 2k or 4k of **additional** EPROM (a total of either 10 k or 12k);

V-D and W-C for either 4k or 6k of **additional** EPROM (a total of either 14k or 16k).

If even more EPROM is required, the same procedure takes place as for additional RAM; links are made to the next available one or two pins of IC5 which have not been used, each pin having a lower number than its predecessor.

2708 EPROM If in table 12a address line A11 is replaced by A1Ø and A12 by A11, the selection order for IC25 . . . IC28 will refer to the 2708 EPROM. Table 15 provides the preferable way in which to wire the first (or only) EPROM section. The required wire links are:

e-f and a-c; P-Q and S-T,

V-W-F.

To increase the 2708 EPROM area above 4k a second RAM/EPROM card is required. The selection information is provided in table 16 and these are the wire links:

e-f and a-c; P-Q and S-T, V-W-E.

Table 12a. The order of selection of IC25 . . . IC28 using links P-R and S-T (2716).

A12 A	A11 B	IC25 k	IC26 l	IC27 m	IC28 n
Ø	Ø	Ø	1	1	1
Ø	1	1	1	Ø	1
1	Ø	1	Ø	1	1
1	1	1	1	1	Ø

Table 12b. The order of selection of IC25 . . . IC28 using links P-T and S-R (2716)

A12 B	A11 A	IC25 k	IC26 l	IC27 m	IC28 n
Ø	Ø	Ø	1	1	1
Ø	1	1	Ø	1	1
1	Ø	1	1	Ø	1
1	1	1	1	1	Ø

Table 13. The decoding system for the (2716) EPROM contained on the first card.

amount of EPROM	IC25	IC26	IC27	IC28
2K	—	—	—	F8...FF
4K	—	—	FØ...F7	F8...FF
6K	—	E8...EF	FØ...F7	F8...FF
8K	EØ...E7	E8...EF	FØ...F7	F8...FF

Table 14. The decoding system for the (2716) EPROM contained on the second card.

amount of EPROM	IC25	IC26	IC27	IC28	
10K	—	—	—	D8...DF	including 8K of Table 13 (first RAM/EPROM-card)
12K	—	—	DØ...D7	D8...DF	
14K	—	C8...CF	DØ...D7	D8...DF	
16K	CØ...C7	C8...CF	DØ...D7	D8...DF	

Table 15. The decoding system for the (2708) EPROM contained on the first card.

amount of EPROM (2708)	IC25	IC26	IC27	IC28
1K	—	—	—	FC...FF
2K	—	—	F8...FB	FC...FF
3K	—	F4...F7	F8...FB	FC...FF
4K	FØ...F3	F4...F7	F8...FB	FC...FF

Table 16. The decoding system for the (2708) EPROM contained on the second card.

amount of EPROM (2708)	IC25	IC26	IC27	IC28	
5K	—	—	—	EC...EF	including 4K on the first card.
6K	—	—	E8...EB	EC...EF	
7K	—	E4...E7	E8...EB	EC...EF	
8K	EØ...E3	E4...E7	E8...EB	EC...EF	

Additional 2708 EPROM is obtained by including another or several more card(s) and then using the D, C, B etc connections to IC5.

What's more . . .

1. Do not forget the wire link L-M!!
2. The RAM/EPROM card must be equipped with a male connector as shown in figure 19a. This should be mounted on the component side of the board. Depending on the number of memory cards installed, the bus board will have to be provided with 1. . . 5 female connectors. This is illustrated in figure 19b. The memory cards are then mounted at right-angles to the bus board.
3. Make sure that the connection points to IC5 are made in the correct order. This is very important!
4. If any of the points V, W, X or Y are not used connect them to a point that is, or to the +5 V supply line.
5. The best way to build up additional EPROM on the memory card is to start at address FFFF (working your way down the address range), because **EPROM must be installed on page FF!** (At least at addresses FFFA . . . FFFF). **The relevant vector information must be stored at locations FFFA . . . FFFF!** This aspect was considered in great detail during the description of the data bus control system on the interface board.

To be able to keep the original EPROM (version D) on the main board of the Junior Computer, the following data must be stored in locations FFFA . . . FFFF of IC28:

NMIL; addresses FFFA data 2F;
NMIH; addresses FFFB data 1F;
RESL; addresses FFFC data 1D;
RESH; addresses FFFD data 1C;
IRQL; addresses FFFE data 32;
IRQH; addresses FFFF data 1F.

Once the Junior Computer is switched on a jump is made to the RESET start routine contained in the original monitor program. The IRQ and NMI jump vectors can be specified in the PIA RAM (page 1A) in the usual manner. Obviously, the operator is free to choose a different solution in connection with his/her system programs. If, however, the reset vector should fail to be specified, the Junior Computer will be as dead as the proverbial doornail when it is switched on! Whatever happens, the NMI and IRQ vector data must be specified to allow an indirect jump to take place to RAM, where the jump vectors can be determined according to the situation. See figure 32 and the relevant text on the subject in chapter 3 of Book 1.

6. By no means expect to be able to remove an EPROM from one socket and simply place it in another. Not only will the start address have changed, but also all the operand data (at least the high order bytes) that refer to the absolute addresses stored in the EPROM. To carry out such an operation the EPROM will have to be re-programmed and the operand data mentioned will have to be completely altered. For these reasons, it is absolutely necessary to have a fully documented program listing for each EPROM. A hex dump alone is not enough, unless the operator has his/her own disassembler routine or investigates the contents of the EPROM 'by hand', which invariably leads to mistakes.
7. When the bus board contains several memory cards, one card can be filled with 2716 EPROMs and another 2708 EPROMs. The same 'start at the bottom' rule applies. It is up to the programmer to select the corresponding wire links. It does not matter so much here if an EPROM area has a 'gap', especially if only part of the available 56k is actually used. It does matter, however, if devices of a different type are mounted on the same board!!
8. Any 4k memory range decoded by IC5 and located on a RAM/EPROM card may be partially filled with RAM or EPROM. If that memory range covers several cards, however, the operator is not permitted to store data in the unused section of a different card. Why not? Well, supposing we have a 4k data block that covers part of one card and part of another, or several others. This does not necessarily mean that the full 4k memory range is involved. Just to recap: as soon as one address in the 4k range is accessed the whole range is selected. If a read operation is involved, where data is retrieved from a single card, the data buffers will be enabled on 'read' for the cards on which the 4k is being selected. This leads to the undesirable situation shown in figure 6.

Conclusion: A 4k address range on a single card does not necessarily have to be used completely, but then the other cards should remain out of the picture altogether. To put this in practical terms: When a bus board contains several RAM/EPROM cards, every link to a particular address decoder (IC5) pin is either used only once or not at all.

9. The links between the various ground connections are all situated on the interface board, around the output connector (4a/c, 16a/c and 32a/c)
10. One RAM/EPROM card may be connected directly to the main board. This is described in Appendix 1.

We have now reached the end of chapter 10 and Junior Computer owners are invited to read chapter 11 and/or chapter 12. Chapter 11 deals with the TAPE MONITOR system program, once the computer is connected to a cassette recorder (or cassette recorders) and the PLL has been calibrated. It is possible to skip this chapter and move directly on to chapter 12 instead (do not pass GO, do not collect £ 200), where further peripheral equipment is discussed along with the various modifications which are needed. This includes the PRINTER MONITOR system routine.

Whichever chapter you read first, it will mean a further journey along the road of discovery regarding computer facilities, for chapter 10 was just the beginning. The journey will not always be an easy one, but at least there is one encouraging thought to help you on your way: at the end of the road there will be a versatile personal computer waiting for you!

11

The cassette interface: a magnetic memory

Storing data on and retrieving data from tape

The combination of cassette interface hardware and Tape Monitor software enables simple transfer of data from the Junior Computer to an ordinary magnetic tape, and vice versa. Basically, what happens here is that some form of load instruction (LDA) or some form of store instruction (STA) is involved. Complete blocks of data (files) or even entire programs can be stored or loaded in one go. The 'memory' in which the data is now stored is not the usual RAM type that suffers from amnesia as soon as the power is switched off, but the 'permanent' type inherent to magnetic tape.

There are a number of advantages to be derived from working with a cassette recorder (or two). For one thing, the programmer can at last start to collect useful programs and even set up a full scale library. All that is then required is to select a particular cassette, slip it into the machine and let the Junior Computer sort out which program file to load into memory. This saves the time consuming job of entering strings of instructions and the accompanying risk of 'typing errors'. All key entry is reduced to an absolute minimum because of five new key functions which replace the 'old' AD . . . PC keys.

What is involved are not exactly 'new' keys, but brand new functions which play a very important role in the 'mechanics' of the cassette interface system. Unfortunately, this means that the relevant keys need to bear yet another set of inscriptions.

Programs can now be entered into a much larger address area running from Ø2ØØ up to Ø7FF without any gaps. This amounts to a total of 1536 bytes which, thanks to the recording power of the cassette interface, only have to be entered once.

Now to unveil the technical mysteries involved in storing data on and retrieving data from cassette tape. It is not as complicated as it may appear at first sight, as readers will find out once they have this chapter 'taped'!

It may seem rather remarkable, when you think about it, that digital signals can be recorded on tape and 'listened to' in exactly the same manner as a piece of Bach, the latest top twenty or the F.A. Cup Final. However, inadequate mortals that we are, we are not able to decipher these signals! The only feasible answer is that the cassette recorder system must possess superhuman powers. This is in fact exactly what happens, as will be seen later when data is converted into specific frequencies which only the recording system (and the computer) can understand.

Before delving too deeply into any of the more technical details, let us return to the various reasons for wanting to use a cassette recorder in the first place. As it was stated previously, storing programs on cassette saves a lot of time and effort. Supposing, for instance, a long program needs to be edited and assembled. The entire program can now be stored on tape before being assembled and then if anything turns out to be wrong, there is no need to re-edit the whole program. What happens now is that the edited version of the program is entered from tape once again, the editor is activated by means of a warm start entry and the program is corrected where necessary. A few more labels may well have to be added, etc. The 'new' situation may then be recorded before the assembly procedure starts and this can be repeated as often as is required. Thus, in this example, the cassette tape acts just like a data bank.

There are, of course, other benefits as well. The horizons of the Junior Computer can be widened considerably, for it is now able to read programs which were not specifically developed for it, such as computer games, etc. The programmer may even be able to join a 6502 'user's club' and exchange programs with his/her fellow members. This would provide a real opportunity to set up a useful **data library!**

The cassette: a magnetic RAM

The benefits described above are available at the reasonable cost of a simple mono cassette recorder and a couple of cassette tapes. If, for example, a C-60 type is used, about 25 minutes worth of recording is available per side (making allowances for gaps in between files and a short space at each end of the tape). This means that with a transmission speed of 50 bytes per second (more about this later) 25 x 60 x 50 = 75.000 bytes (or 73 kilo bytes) can be stored — and that is just on one side! It should be enough to start with . . .

How is the data actually stored on tape?

Figure 1 looks rather like a train carriage with a series of compartments in which different information is 'seated'. The data does not necessarily belong to a complete program, it may also constitute a separate routine, a table or a length of text, etc. In any case, however, a **data block** is involved, which is sent to the tape recorder during **data transmission.**

Now let us examine the configuration shown in figure 1:

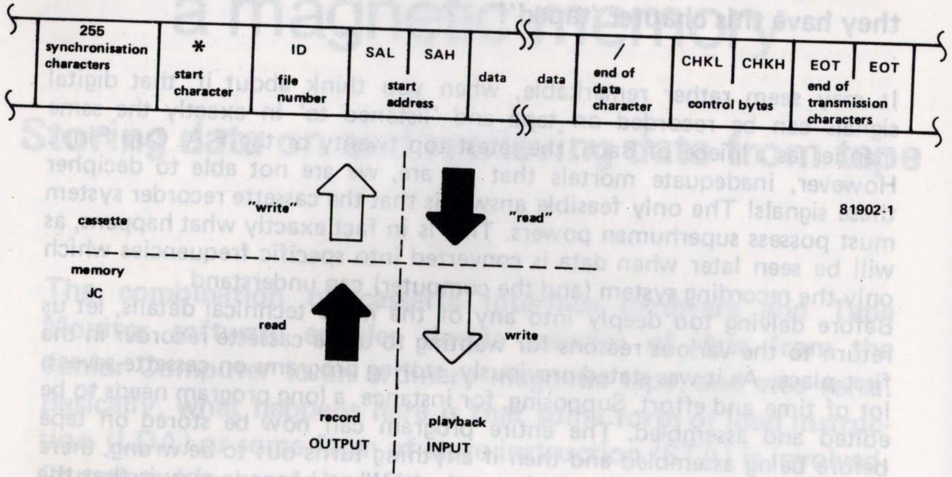


Figure 1. The various 'compartments' belonging to a data transmission.

1. Synchronisation characters.

Considering the tape from left to right, the first 'compartment' consists of 255 synchronisation characters. These determine the actual start of a data block and enable the Junior Computer to distinguish data from other information which may have been recorded previously. In other words, the computer will ignore the first babblings of your new baby, your brother's trumpet voluntary or such remarks as 'this is my first program on cassette'. . .

Since a total of 73k bytes is available on one side of a C-60 cassette tape, there is ample room for several data blocks and/or programs on the same cassette. Therefore, a single tape will contain a number of data transmissions with short breaks in between them.

What, in fact, is a synchronisation character? Basically, what is meant is an **ASCII coded character**, for data is always coded in the ASCII format — 8 bit words — when it is recorded on tape. The most significant bit is reserved for special purposes and in this particular instance will always be logic zero. The various bits in the ASCII byte are stored one after the other, serially. Serial data transmission will be discussed at length in chapter 12. The synchronisation character is 16 in hexadecimal. Thus a data transmission will start with ØØØ1Ø11Ø which is repeated 254 times.

2. The start character '*'

Once the start of a data transmission is known, the start of the data itself needs to be announced. This is accomplished by the character '*' (hexadecimal 2A in ASCII).

3. The file number 'ID'

Obviously, if a great number of data blocks are to be stored on one (side of a) tape, each individual block will have to have its own identity to avoid confusion. Thus, ID really stands for ID in this instance! A total of 254 different file numbers are possible. This includes all the various values between Ø1 and FE. Values ØØ and FF are reserved for a specific task when data is read from tape. Each data block must be preceded by its ID.

4. The low order address byte, SAL

This corresponds to the low order byte of the first address location pertaining to the program or data block that is to be stored on cassette.

5. The high order address byte, SAH

This corresponds to the high order byte of the first address location pertaining to the program or data block that is to be stored on cassette. Each address byte is transcribed into two ASCII characters, that is to say, one ASCII character per nibble. The data bytes are then stored on tape as a series of 16 bits. The write operation begins with the start address (contents of SAH and SAL) and ends with the address minus one stored in locations EAH and EAL. As figure 1 shows, the end address (EA) does not have a 'compartment' of its own, but is determined by the start address and the length of the data block.

Some readers may wonder why we chose to call the last address EA instead of the more logical LA. Well, this is because the procedure had to be compatible with that of the KIM computer. Obviously, it is important to ensure that the contents of SA are always less than those of EA.

Why are SA and EA necessary? The computer must know where (at which address) to start reading (storing on tape) or writing (reading from tape) the information from/to RAM. It is possible to select a different start address when reading data from tape (ID = FF), but this may not be a particularly good idea. Remember that a program (= data block) contains absolute address operands which will have to be modified either partially or completely if a new start address is chosen (only the high order address bytes have to be altered during a page shift), in other words, if the data is to be stored in a new memory area.

6. The actual data block

Since each data byte consists of two nibbles, each of which is stored on tape as an ASCII coded character, 16 bits per data byte are sent, bit by bit, to the tape. The data block contains all the information belonging to locations SA . . . EA (see figure 2). In principle, a data block may be any

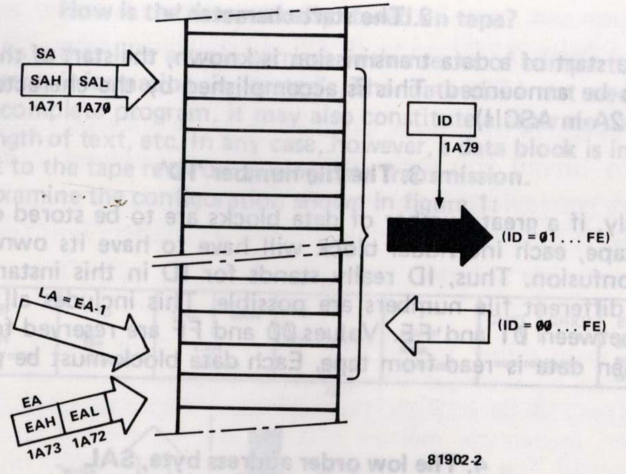


Figure 2. A data block containing the parameters involved during transmission.

size. Both RAM and EPROM data may be stored on tape (= read from memory), but only RAM can be filled with data contained on tape, as an EPROM can only be written to once during the initial programming.

7. The end of data character '/'

This character (ASCII code 2F) informs the Junior Computer when all the data in the block has been read from tape (when data is stored on tape, the end of data character is recorded when the contents of the address pointer, POINTH and POINTL, coincide with those of EA).

8. The control bytes CHKH and CHKL

These two bytes ensure that nothing was mislaid or misread during a recording. A cassette recorder has a special gift for distorting information, in spite of the improved PLL (see chapter 10). It is therefore absolutely necessary to check whether one or more bits happened to be viciously mutilated in some invisible manner, the best method being to count them like a shepherd counts his sheep.

At the beginning of a write operation, the CHKH and CHKL bytes are made equal to zero. From SAL on, the bytes are all added to each other before being translated into the ASCII code. The file number (ID) is not taken into account here. Each time the contents of CHKL reach FF, the contents of CHKH are incremented by one, and when this also reaches FF, both are reset to zero without causing any complications. When data is read from tape, the same procedure is used.

All that has to be done then is to compare the result of the two counts: the number of bytes during the read operation must be the same as those during the write operation, and vice versa. If so, there is reason to believe (and hope!) that the data transmission has been carried out succesfully. If not, the tape will have to be re-loaded and maybe even tidied up here and there. We could of course provide a whole chapter of error statistics and other theoretical titbits . . ., but at this stage it is better for readers to cross that bridge when they come to it.

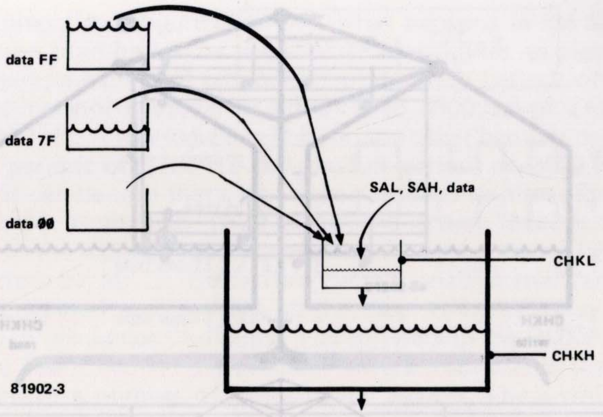


Figure 3. The 'beer barrels' illustrate the purpose of the checkbytes CHKH and CHKL.

The check system can be illustrated by a pair of beer barrels (see figure 3). Location CHKL is represented by a small beer barrel and CHKH by one that is 256 times (= FF) larger. The beer represents the flow of data. If the data amounts to FF, the small barrel will be filled to the brim. If, however, the data is ØØ, the barrel will be empty. As soon as the small barrel threatens to overflow, the entire contents will flow out of the bottom of the barrel. The same will happen to the large barrel when this is also full of beer, the entire contents will disappear. Once all the beer/data has stopped flowing (assuming, of course, that none was drunk on the way!), the final levels may be compared.

How the balance of the two levels is drawn up is illustrated in figure 4. If the two large and the two small beer barrels are equally full, everything will be all right. If one of them is slightly emptier, you can be sure that someone has been having a few 'tipples' on the side.

Another way of looking at the check system is to compare it with a bank. It is in the bank's common interest (and ours!) to see that people's money is well looked after. It would therefore be disheartening, to say the least, for the employees to discover one morning that one of the clients had remained in the building after closing and has subsequently absconded with all the cash. This can be avoided quite simply by counting all the clients who come in and go out during the day and then compare the two numbers. If they do not tally, the bank manager may well have reason to worry. It could mean that the system has forgotten how to count correctly, or that two persons passed the detector at exactly the same time and were therefore registered as only one, or even that a client cashed in on his/her pools winnings and was shown out discretely through a door at the back. If the two latter cases mentioned happened to coincide they would auto-

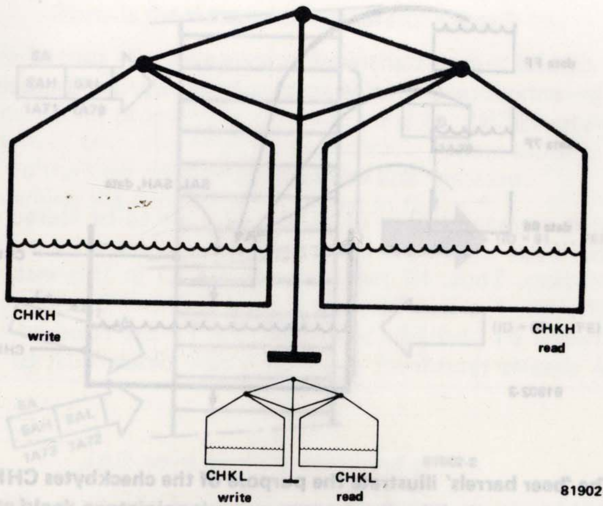


Figure 4. The 'scales' illustrate the comparison of the two values for CHKH and CHKL.

matically cancel each other out and so would escape notice altogether. Hardly ideal!

As far as the check bytes CHKH and CHKL are concerned, however, counting the bytes by adding them is, believe it or not, a perfectly foolproof method, in spite of the peculiar circumstances which might arise.

9. Two EOT (End Of Transmission) characters

These 'close the gate behind the sheep' to indicate that a complete data block has been transmitted (ASCII code: Ø4). That just about covers all the 'compartments' in figure 1. As you probably noticed, it was quite a long train of thought. Apart from the fact that 255 synchronisation characters are involved instead of 100, the data transmission format is identical to that of the KIM computer.

How to store data on tape

Digital music in the form of three notes per bit

As we mentioned earlier, data is transmitted serially, bit by bit. The question is: How are the bits stored on tape? A logic one is commonly associated with a 'high' level and a logic zero with a 'low' level. Consequently, a logic one bit could be translated into a high frequency and a logic zero into a low frequency. A flow of data bits would then be stored on tape as a series of high and low tones. Such a system is the 'Kansas City Standard' where the two tones are 2400 Hz and 1200 Hz respectively. Things are slightly more complicated in the case of the Junior Computer, where both bits are translated into a high frequency tone followed by a low frequency tone. The difference between a logic one and a logic zero

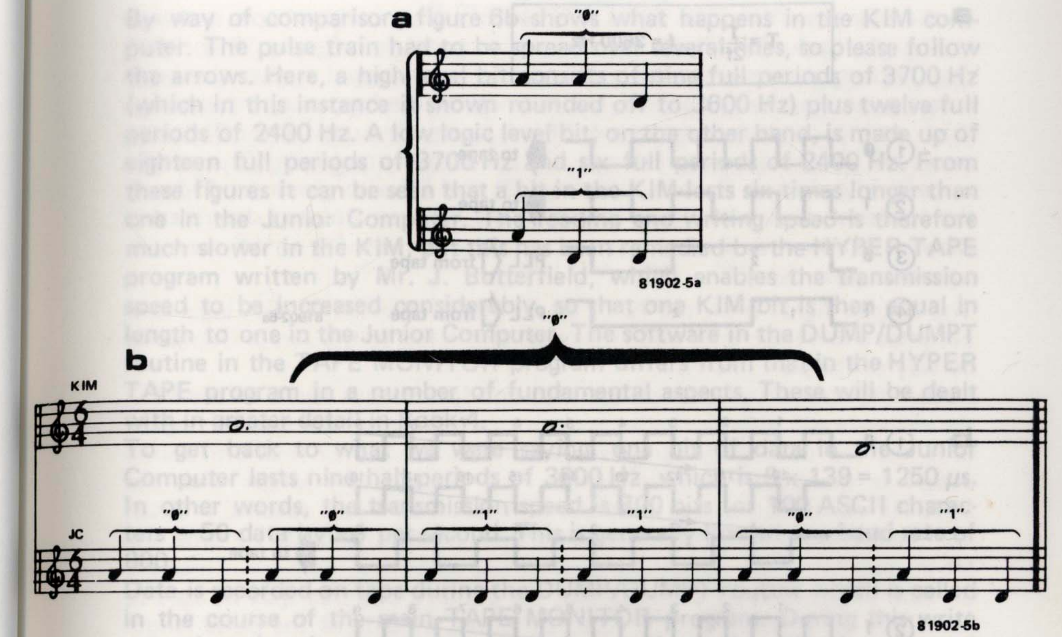


Figure 5. Say it with music: a logic zero bit can be represented by two high notes followed by a single low note, whereas a logic one bit can be represented by a single high note followed by two low notes. The 'music' is recorded and played back by the Junior Computer at a rate of almost six times the speed of the KIM (figure 5b).

lies in the length of the high frequency tone with respect to the length of the low frequency tone, the total length of each pair being the same. In the case of a logic zero, the high frequency tone lasts twice as long as the low frequency tone, whereas the exact opposite is true for a logic one.

The time ratio will therefore be either 2:1 or 1:2. Since one plus two equals three, a bit can be represented by three 'musical' notes. This is illustrated in figure 5a, where the 'melody' for the logic zero bit consists of two high notes followed by one low note, and the 'melody' for the logic one bit is made up of a single high note followed by a pair of low notes.

Fortunately, the Junior Computer does not need a piano to write 'music' and record it on tape. The two frequencies involved are 3600 Hz and 2400 Hz. These, we assure you, are well outside the range of any soprano! The various pulse trains are shown in figure 6a. A low logic level bit consists of six half-periods of 3600 Hz and two half-periods of 2400 Hz (pulse train (1) in figure 6a). A low logic level bit is comprised of three half-periods of 3600 Hz and four half-periods of 2400 Hz (pulse train (2) in figure 6a). The total length of the train remains unchanged, irrespective of whether the logic level is high or low. (This can be expressed as 9T where T equals the duration of one half-period of 3600 Hz). Furthermore, the train will always start with the higher frequency. The duration ratio for the high and low frequencies is either 2:1 or 1:2. In the musical example in figure 5a each note lasts 3T.

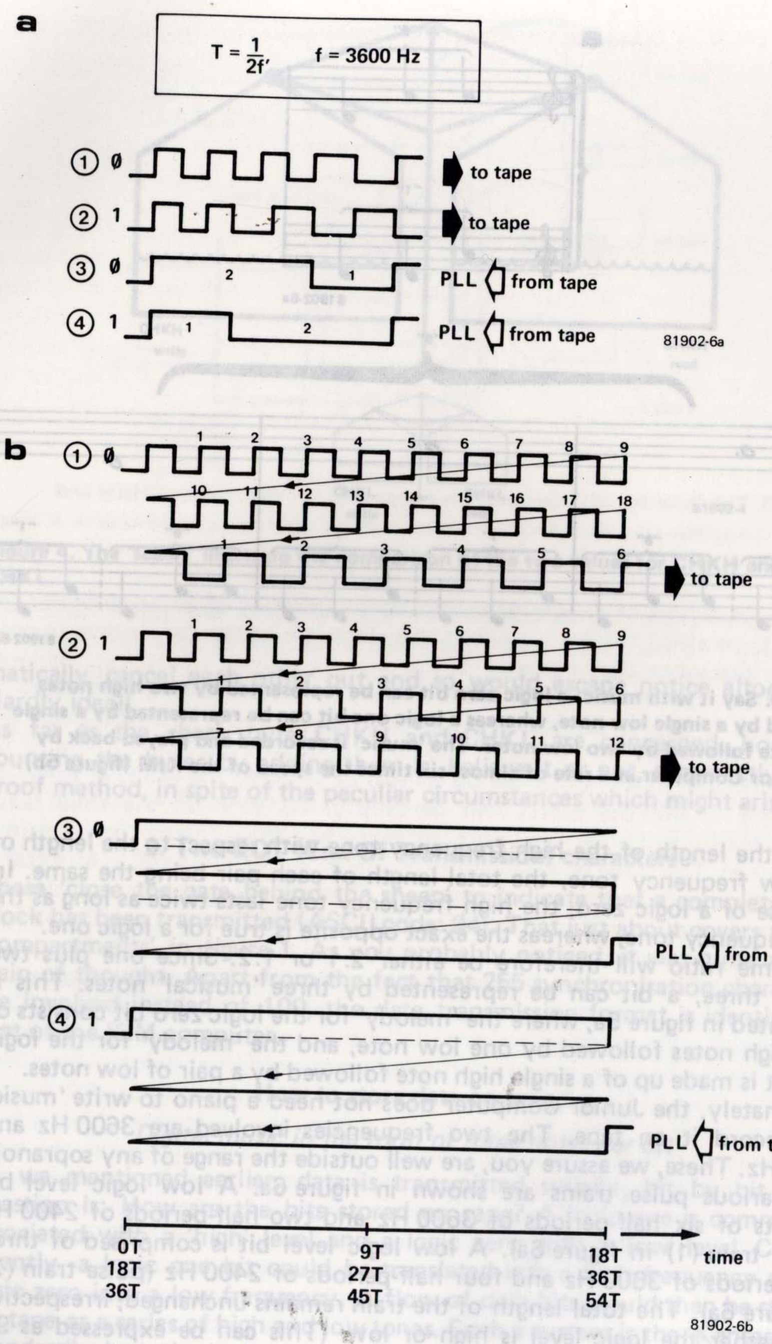


Figure 6. The signals transmitted to tape during the write operation, (1) and (2), for the Junior Computer (figure 6a) are much shorter than those for the KIM (figure 6b). Consequently, the output signals from the PLL in the playback mode, (3) and (4), are also of much shorter duration in the Junior Computer.

By way of comparison, figure 6b shows what happens in the KIM computer. The pulse train had to be spread over several lines, so please follow the arrows. Here, a high level bit consists of nine full periods of 3700 Hz (which in this instance is shown rounded off to 3600 Hz) plus twelve full periods of 2400 Hz. A low logic level bit, on the other hand, is made up of eighteen full periods of 3700 Hz and six full periods of 2400 Hz. From these figures it can be seen that a bit in the KIM lasts six times longer than one in the Junior Computer. The reading and writing speed is therefore much slower in the KIM, but this has been remedied by the HYPER TAPE program written by Mr. J. Butterfield, which enables the transmission speed to be increased considerably, so that one KIM bit is then equal in length to one in the Junior Computer. The software in the DUMP/DUMPT routine in the TAPE MONITOR program differs from that in the HYPER TAPE program in a number of fundamental aspects. These will be dealt with in greater detail in Book 4.

To get back to what we were saying, one bit of data in the Junior Computer lasts nine half-periods of 3600 Hz, which is 9 x 139 = 1250 μs. In other words, the transmission speed is 800 bits (or 100 ASCII characters = 50 data bytes) per second. This is generally known as a **baud rate** of 800.

Data is recorded on tape during the DUMP/DUMPT routine which is called in the course of the main TAPE MONITOR program. During this write operation the six seven-segment displays will remain unlit, but the red (OUTPUT) LED will be on. The parameters for a successful operation are:

a. Indicate the program number, ID (Ø1 . . . FE, ØØ and FF serve a special purpose, which we shall come back to later).

b. Indicate the start address SA.

c. Indicate the end address EA (one location higher than the last address, LA, of the data block). The data block is read from memory starting with the first address, SA, and ending with the last address, LA. This is accomplished with the aid of the well known address pointer POINT, which is used to display an address and its contents.

At the end of the DUMP/DUMPT routine, the contents of POINT should equal the contents of EA. This can be verified by depressing the RST key. The display will then show the address EA and its contents.

How to read data from tape

From bits on tape to bytes in memory

Returning to figure 6a, the data bits are recorded on tape either as two high frequency tones followed by a low frequency tone (logic zero) or as one high frequency tone followed by two low frequency tones (logic one). When the information is read from tape, the tones have to be translated back into bits and reorganised into ASCII bytes, which in turn have to be converted into data bytes. The latter must be stored in memory, in serial format, from the start address on.

Let us first reconstruct the bits. During the read operation, the recorded signals (1) and (2) in figure 6a feature a symmetrical, rectangular waveform (see also figure 17). The record head of the cassette machine will, however, cause them to lose this rectangular form. Due to the effects of the low pass filter, the signal will become sinusoidal in shape, as shown at the playback INPUT of the Junior Computer (see figures 18 and 19). The important factor as far as the PLL is concerned is that the zero-crossing information is not lost. In other words, the frequency information (high and low tones) is preserved. This is quite adequate for the PLL, as a result of which, drawings (1) and (2) are transformed into (3) and (4), respectively. As can be seen, drawings (3) and (4) also give a good idea of the 'melody' shown in figure 5a. When the PLL control was described in chapter 10, the transition from (1) to (3) and from (2) to (4) was discussed in great detail.

The output signal from the PLL, (3) or (4), is inverted (a high logic level becomes a low logic level and vice versa) and is fed to the Junior Computer by way of the port line PB7.

According to the duration of the high and low output levels of the PLL, the software will filter out either high or low logic bits from the PLL signal. It is not the durations themselves that matter, but their relation to each other: is the 3600 Hz period (high PLL output) any longer than the 2400 Hz period (low PLL output), or vice versa? Their specific T values are irrelevant. If the 3600 Hz tone lasts longer than the 2400 Hz tone, the bit in question will be logic zero, otherwise it will be logic one. Since the durations themselves are immaterial, the Junior Computer may be used to read back data taped by the KIM computer, in spite of the pulse length differences between the two systems. Consequently, signals (3) and (4) in figure 6a are identical to those in figure 6b, even though the latter signals are six times longer than the former. This is a great advantage, even for readers who do not have occasion to use the KIM, as it means that variations in the recording and playback speeds (nominally 4.75 cm per second) will not affect the transmission quality — there will be no 'flutter', etc. In other words, it does not matter which type of tape recorder is employed and it is possible to record on one machine and playback on another. This is because the 2:1 (low logic level) and 1:2 (high logic level) ratios are too wide apart to cause confusion.

Ratios of 4:3 (instead of 2:1), due to temporary interference, or 3:4 (instead of 1:2) do not cause the read software any problems, as even then it manages to detect the correct bit value. As figure 5a shows, this is quite something, for if the diagram containing two high tones and one low tone differs greatly from its opposite number, the drawing containing four high tones and three low tones is almost identical to the one containing three high tones and four low tones!

In chapter 10 the question of PLL jitter came up in connection with unauthorised, premature or indefinite changes in the logic level of the output. PLL jitter is not taken into account during a read operation. In actual fact, both the high frequency tone durations and the low frequency tone durations are trimmed down a little. This means that the contrast between the 1:2 and 2:1 ratios is narrowed down slightly, but not enough to cause confusion. PLL jitter may, however, cause problems when attempts are made to speed up the read and write operations. This is because the bit times are shortened somewhat. So do not place too much trust in the high speed cassette interfaces currently advertised, the majority only lead to data readings full of errors!

Subroutine RDTAPE

Data is read from the cassette tape with the aid of the subroutine RDTAPE, which is called during the TAPE MONITOR program. The two right-hand displays of the Junior Computer conveniently indicate what is going on during the read process. The remaining four will stay unlit. The two right-hand displays light in three specific configurations (see figure 7 and 8).

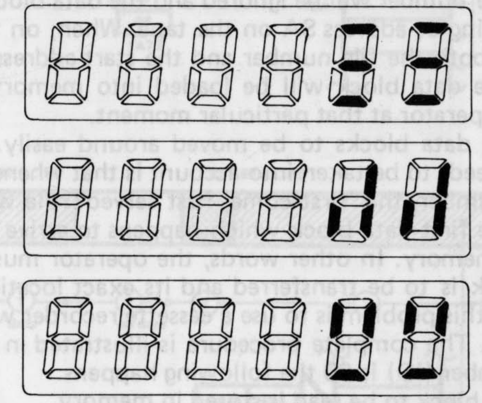


Figure 7. When a data block (file) is being searched for, detected and loaded from cassette, the two right-hand displays of the Junior Computer will indicate the status. This is not the case with the KIM computer.

The first drawing in figure 7 applies when:

a. the tape passing the playback head (cassette is in the 'play' mode) contains no digestible data (space between two data blocks, unrecorded tape, Beethoven's 5th, etc.). The two displays will be seen to flicker during this period.

b. the tape contains a data block which is being read, but the beginning is missing, or the ID number does not correspond to the one specified. In this instance the two displays will not flash, but will remain lit constantly.

The second situation occurs when the computer is in the synchronisation phase. In other words, it is reading the synchronisation characters preceding a data block. The reading may well not be quite perfect with regard to the initial few characters, so the display pattern in the drawing will flash for about one second before becoming stable. There are 255 synchronisation characters on the tape. The computer takes about 2.5 seconds to read these ASCII bytes. The Junior Computer is able to detect the beginning of a data block as soon as it has read an uninterrupted sequence of ten synchronisation characters. Since there are 255 of these altogether, the Junior Computer has an excellent chance (in fact, at least 20 possibilities) of making an unambiguous detection. The KIM on the other hand, only has 100 synchronistaion characters and so things are far more likely to go wrong here.

The third situation shown in figure 7 occurs when the ID number specified by the operator has been found and stored in the memory of the Junior Computer.

Before data recorded on tape can be read, that is before the jump to the RDTAPE subroutine, an identification number must be specified. Even though a tape may contain up to 254 different data blocks, entering the number of one of them is sufficient for the computer to trace the correct file. If either ØØ or FF is entered as an ID number, the computer will load the first data block that happens to come along. Moreover, when ØØ is specified, the file number will be ignored and the data block will be stored in memory starting at address SA on the tape. When, on the other hand, FF is specified, both the file number and the start address are ignored. In this instance, the data block will be loaded into memory at an address selected by the operator at that particular moment.

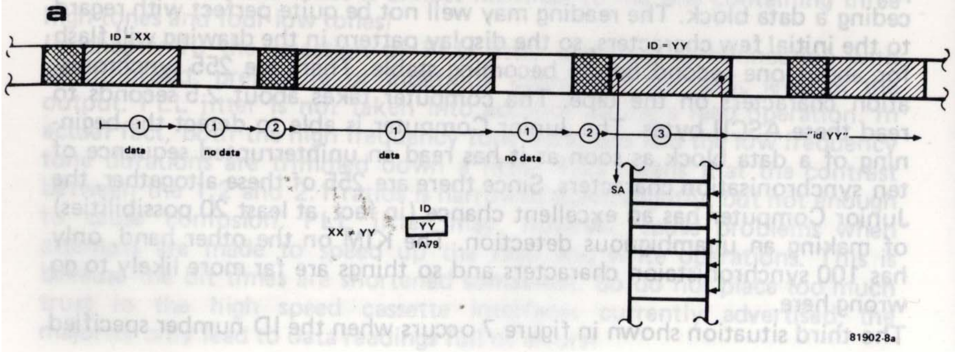
This enables the data blocks to be moved around easily. The only consideration that needs to be taken into account is that whenever ØØ or FF is used as an ID number, the 'first come, first served' rule will be enforced, meaning that the first data block which happens to arrive in due time will be loaded into memory. In other words, the operator must know exactly which data block is to be transferred and its exact location on the tape. One solution to this problem is to use a cassette recorder which has a 'tape counter' built-in. The complete procedure is illustrated in figure 8a. When the program number (ID) is ØØ the following happens:

- The first data block to be read is stored in memory.
- The start address, SA, is the same as that stored on tape.
- The program number which was recorded on tape is therefore ignored.
- See figure 8b.

When the program number (ID) is FF, the following happens:

- The first data block to be read is stored in memory.
- The start address, SA, is the same as that previously entered by the operator.
- The program number which was recorded on tape is therefore ignored.
- The start address recorded on tape is also ignored.
- See figure 8c.

This is how data blocks can be stored in memory without losing the start address recorded on tape. Programs can now also be relocated in memory. This is why a tape counter is essential. In some cases, for instance, moving a data block may mean that absolute address operands have to be modified, either partially or completely. Such things, however, do not yet exist in programs generated by the editor. The label information is still included in such programs. By using the identifier FF, and a start address chosen by the operator, it is possible to shift a data block to any memory area and assemble it with the assembler.



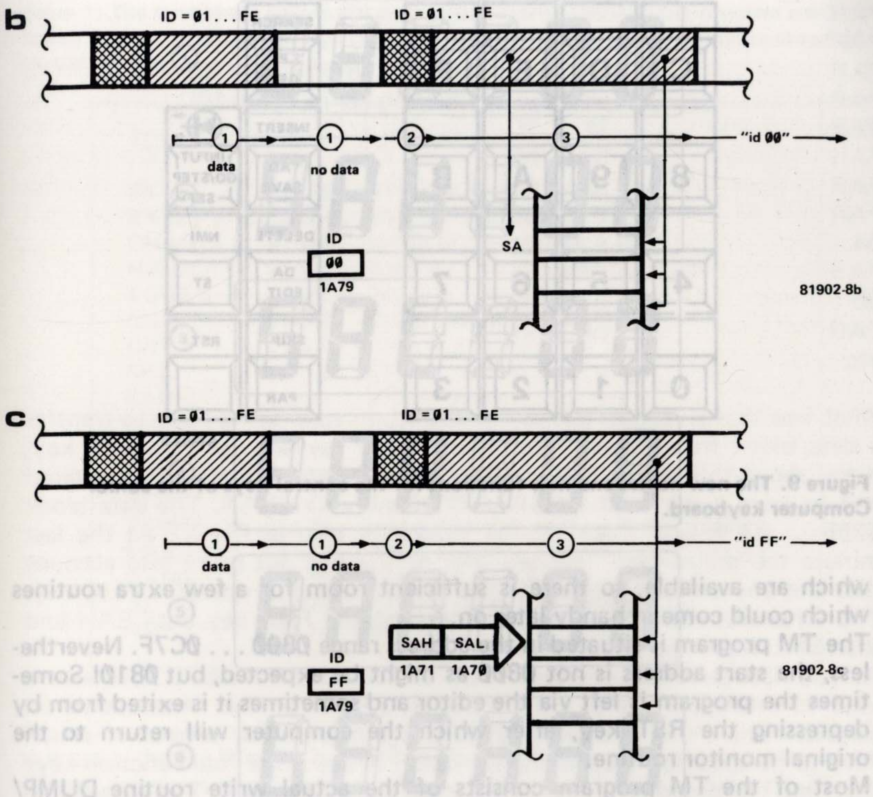


Figure 8. The search for a data block when the identifier (ID) equals Ø1 . . . FE (figure 8a), when ID = ØØ (figure 8b) and when ID = FF (figure 8c).

Data management

The TAPE MONITOR program

The TAPE MONITOR system program (which we will refer to as TM from now on) is designed to fulfil all the wishes of the operator with regard to writing data from the Junior Computer onto tape and reading data stored on tape into the Junior Computer memory. This program is resident in a 2716 EPROM and is therefore available as soon as the Junior Computer is switched on. The EPROM can be programmed using the hexadecimal dump printed in Appendix 4 at the back of this book. Although it consists of more than 1024 bytes, it by no means occupies the full 2048 bytes

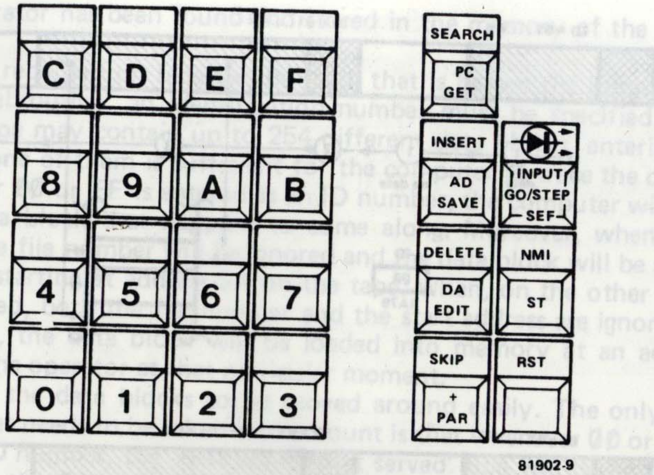


Figure 9. The new names and new functions for the control keys of the Junior Computer keyboard.

which are available, so there is sufficient room for a few extra routines which could come in handy later on.
The TM program is situated in the address range Ø8ØØ . . . ØC7F. Nevertheless, the start address is not Ø8ØØ as might be expected, but Ø81Ø! Sometimes the program is left via the editor and sometimes it is exited from by depressing the RST key, after which the computer will return to the original monitor routine.
Most of the TM program consists of the actual write routine DUMP/DUMPT and the actual read routine RDTAPE. The rest of the program includes software which, on the basis of the new functions given to the keys AD . . . PC, enables various parameters to be set before the data is actually stored on or retrieved from tape. The 'new' keyboard is shown in figure 9. Let us now examine each 'new' function in turn.

1. The PAR key (the 'old' +/SKIP key)

Setting parameters beforehand

Let us start by taking a 'key' example:

Depress:	**The display shows:**
RST Ø 8 1 Ø GO	id ØØ (situation (1) in figure 10)
2C	id 2C
PAR	SAH Ø2 (situation (2) in figure 10)
Ø2	SAH Ø2
PAR	SAL ØØ (situation (3) in figure 10)
PAR	EAH ØØ (situation (4) in figure 10)
Ø3	EAH Ø3
PAR	EAL ØØ (situation (5) in figure 10)
FF	EAL FF

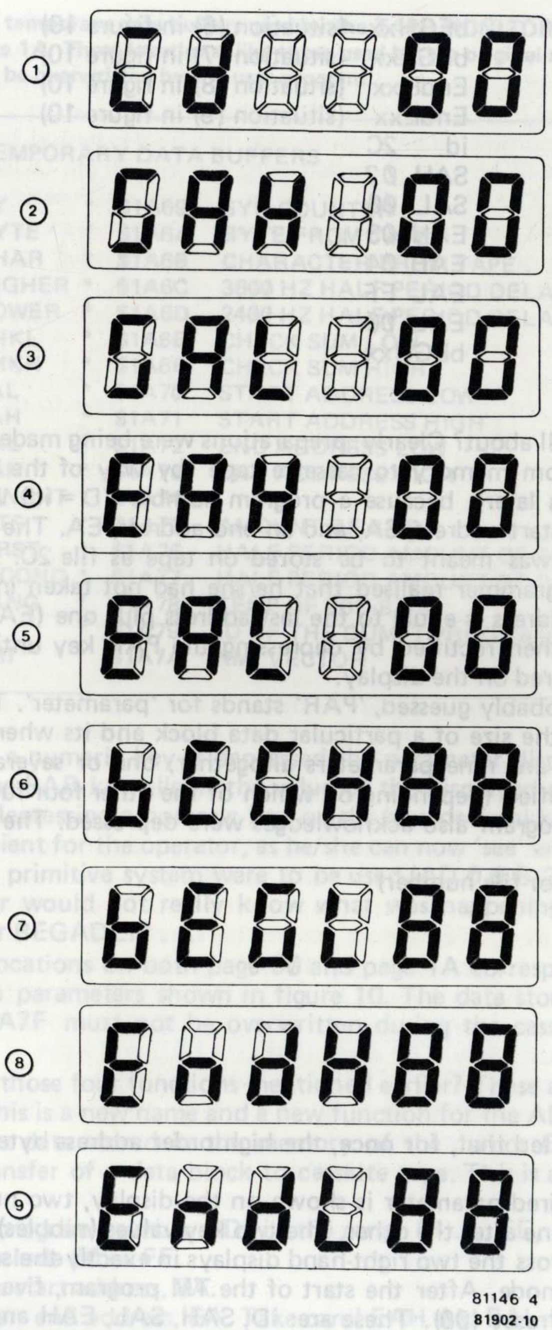


Figure 10. The nine parameters which play an important role when data is being transferred to or from cassette can all be displayed during the TAPE MONITOR program with the aid of the PAR key (= +/SKIP key).

```
PAR     bEGHxx  (situation (6) in figure 10)
PAR     bEGLxx  (situation (7) in figure 10)
PAR     EndHxx  (situation (8) in figure 10)
PAR     EndLxx  (situation (9) in figure 10)
PAR     id   2C
PAR     SAH  Ø2
PAR     SAL  ØØ
PAR     EAH  Ø3
Ø4      EAH  Ø4
PAR     EAL  FF
ØØ      EAL  ØØ
PAR     bEGHxx
etc.
```

What was that all about? Clearly, preparations were being made to transfer a data block from memory to cassette tape (by way of the SAVE key, more about this later), because a program number (ID = id) was entered and so were a start address, SA, and an end address, EA. The data block Ø2ØØ . . . Ø3FF was meant to be stored on tape as file 2C. At the last minute the programmer realised that he/she had not taken into account that the end address is equal to the last address plus one (EA = LA + 1). The error was then rectified by depressing the PAR key until EAH and then EAL appeared on the display.

As you have probably guessed, 'PAR' stands for 'parameter'. This term is used to define the size of a particular data block and its whereabouts on cassette. There are nine parameters altogether, one or several of which have to be specified (depending on which of the other four function keys that the TM program also acknowledges were depressed. The parameters are as follows:

- ID (program or file number)
- SAH
- SAL
- EAH
- EAL
- BEG(AD)H
- BEG(AD)L
- END(AD)H
- END(AD)L

It should be noted that, for once, the high order address byte is specified first.

When the required parameter is shown on the display, two numeric keys are depressed, one after the other. The two key values (nibbles) move from right to left across the two right-hand displays in exactly the same manner as in the DA mode. After the start of the TM program, five of the nine parameters are reset (ØØ). These are: ID, SAH, SAL, EAH and EAL. The other four parameters either have a random value, if the Junior Computer was switched on just before the TM program was called, or the same value as when the computer was in the editor mode. Situations (6) . . (9) in figure 10 show the data '88' to illustrate the 'don't care' condition (all the segments lit).

Table 1. The temporary data buffers used by the TAPE MONITOR program are stored in page 1A. These locations (like those used by the original monitor program) should never be overwritten by the user program.

TEMPORARY DATA BUFFERS

SY	*	$1A69	SYN COUNTER
BYTE	*	$1A6A	BYTE FROM TAPE
CHAR	*	$1A6B	CHARACTER FROM TAPE
HIGHER	*	$1A6C	36ØØ HZ HALF PERIOD DELAY
LOWER	*	$1A6D	24ØØ HZ HALF PERIOD DELAY
CHKL	*	$1A6E	CHECK SUM LOW
CHKH	*	$1A6F	CHECK SUM HIGH
SAL	*	$1A7Ø	START ADDRESS LOW
SAH	*	$1A71	START ADDRESS HIGH
EAL	*	$1A72	END ADDRESS LOW
EAH	*	$1A73	END ADDRESS HIGH
SYNCNT	*	$1A74	
BITS	*	$1A75	AMOUNT OF BITS
FIRST	*	$1A76	HALF PERIOD AMOUNT OF 36ØØ Hz
SECOND	*	$1A77	HALF PERIOD AMOUNT OF 24ØØ HZ
GANG	*	$1A78	TEMP OF PBD-BITS
ID	*	$1A79	ID OF THE DUMPT PROGRAM
NMI	*	$1A7A	NMI VECTOR

As soon as a numeric key is depressed the parameter display changes. In this way, the PAR key allows the value of the displayed parameter to be altered by depressing a numeric key on the hexadecimal keyboard. This is very convenient for the operator, as he/she can now 'see' what is going on. If the more primitive system were to be used (AD Ø Ø E 2 DA x x + y y) the operator would not really know what was happening: ØØE2, is that BEGADH or BEGADL??

N.B. Nine locations on both page ØØ and page 1A correspond to the nine initialisation parameters shown in figure 10. The data stored in locations 1A69 . . . 1A7F must not be overwritten during the cassette read/write operations!

What about those four functions mentioned earlier? These are:

1. **SAVE:** this is a new name and a new function for the AD/INSERT key. It enables data stored in the memory of the Junior Computer to be saved by transfer of a data block to cassette tape. This is accomplished as follows:

- Enter a program number (ID) in the range Ø1 . . . FE, in other words, **do not include ØØ** or FF.
- Indicate a start address, SA.
- Indicate an end address, EA. Take care! EAH and EAL form an address which is one place behind the last address of the data block concerned.

Thus, if the end of the data block is Ø3FF, EAH = Ø4 and EAL = ØØ.

- Find an empty section on the cassette. To ensure that no 'old' data will be overwritten, check by means of the internal speaker whether any 3600 Hz or 2400 Hz signals can be heard.

- Make a note of the current contents of the tape counter, the ID, the SA and the EA. If you like, you can record these comments as well.
- Depress 'record' and 'play' on the cassette player simultaneously. Make sure the record level is adequate. Start the tape.
- Record your comments first or depress the SAVE key at once. A data transmission will now be recorded with the aid of the DUMP/DUMPT routine. The red (OUTPUT) LED will light. The six seven-segment displays remain unlit. Wait until the Junior Computer reports back 'id xx', where xx stands for the program number you just stored on tape.
- Stop the cassette recorder.

That covers the practical side of a data recording. Readers with cassette recorders featuring remote control facilities will find that the cassette starts automatically after SAVE is depressed (provided, of course, the recorder is set in the 'record' mode!). They will then have to switch off the remote control if they wish to add any comments. As soon as the recording has finished and RST is depressed, the end of file address, EA, will appear on the display. This will be the start address, SA, of the next data block.

2. **The GET key:** a new name and a new function for the PC/SEARCH key. When this key is depressed, the Junior Computer reads a particular data block stored on cassette tape into memory. This involves the subroutine RDTAPE.

The following steps are taken:

- Indicate the required program number in the range Ø1 . . . FE. If this is omitted the ID will be taken from the former contents of location ID (situation (1) in figure 10).
- If ID = FF, indicate a start address, SA. Forgetting to do this will cause the old contents of SAH and SAL (situations (2) and (3) in figure 10) to be used for SA. At the start of the TM program, SAH and SAL will be reset (ØØ), so that the start address will be ØØØØ.
- Place the cassette containing the required data block into the recorder. Where ID = Ø1 . . . FE it does not matter whether the exact location of the data block is known, as long as the tape is started early enough. Where ID = ØØ or FF, the tape must be prepared beforehand, as the first data block to arrive will be entered into memory in these instances.
- Depress the GET key. The green (INPUT) LED will light.
- Set the cassette recorder to 'play' and start the tape.

Now, the situations belonging to figure 7 will be displayed. When situation (3) appears, the data block in question has been stored in memory.

Once the file has been loaded, the Junior Computer will inform the operator by displaying 'id xx', where xx stands for the program number that was indicated previously – provided the data has been read in correctly. See the end of this chapter.

- Stop the cassette recorder.

This is the normal procedure, that is, if no mistakes were detected during the checksum calculation. If, however, there were some errors, the computer will not display the text 'id xx'.

Once a data block has been loaded the various parameters can be examined with the aid of the PAR key. Only the contents of ID and (if ID = FF) those of SAH and SAL are relevant. So do not expect to find the end address of the data block at locations EAH and EAL (this only applies when data is written to tape).

N.B. If the RST key is depressed (return to the original monitor) immediately after a file has been read into memory, the appropriate end address, EA (contents of SA plus the size of the data block), will appear on the display. That is, if an ID of FF was not involved.

If the chosen ID was FF, something else will also happen. The address pointer, POINT, which was used during the write operation is decremented by one. Next, the contents of SAH and SAL are made equal to the new contents of POINTH and POINTL respectively. Thus, depressing the RST key would bring the last address, LA, of the data block to the display.

The above allows edited programs to be placed one after the other without loss of space (the EOF characters of the previous data block are overwritten). Bearing this in mind, if any data blocks are to be placed one after the other using the identifier FF, the end address (EAL) will have to be modified (incremented by one) and, if necessary, so will EAH, before a new data block is loaded (otherwise, the last address of the first data block will be overwritten by the first address of the new block!).

This is illustrated in figure 11. Using a start address, SA, that is equal to BEGAD and an end address that is the same as CEND, an edited program is loaded onto tape (see the section on the SEF key). The current end address pointer, CEND, will be pointing to the first vacant memory location. This is one location behind the EOF character 77. The EOF characters will have to be overwritten to allow several edited data blocks to be stored one after the other without loss of space. Thus, one memory location must overlap, which is done by decrementing POINT until it equals the new start address.

3. **The EDIT key:** this attributes a new function to the DA/DELETE key. In actual fact, the function concerned is not entirely new, as activating EDIT is equivalent to entering AD 1 C B 5 GO, leading to a cold start of

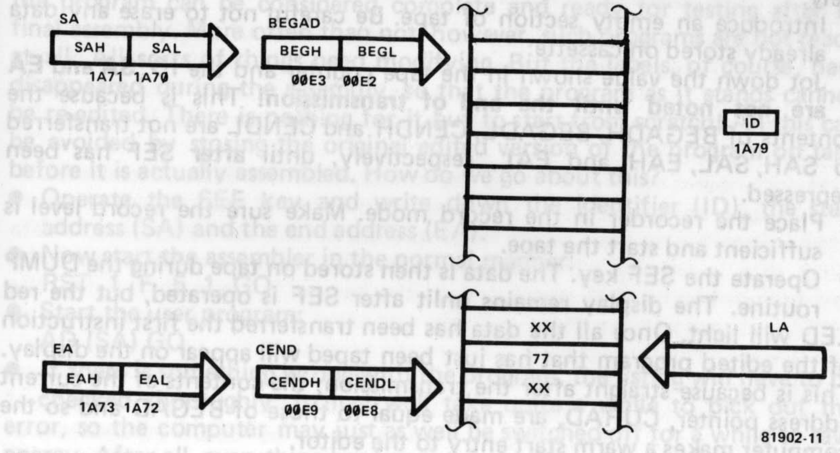


Figure 11. The same situation as figure 2, only now an edited program is involved in the data transmission. The data is transferred to cassette after operating the SEF key and is reloaded by depressing the GET key.

the editor. Prior to this, BEGAD and ENDAD are loaded by means of the PAR key (operate PAR until BEG(AD)H is found, enter the correct data, operate PAR once more and enter the required data into BEG(AD)L, etc. – see points (6) . . . (9) in figure 10). As you know, a cold entry to the editor causes '77' to appear on the displays. This is exactly what happens when EDIT is operated as well, so that it may well seem superfluous. Is this really necessary? The answer is yes, as will be gathered from the following.

4. **The SEF key:** this is a new function given to the GO/INPUT key. It has a special task to fulfil just before the TM program is started. SEF stands for Save Edited File; in other words, a data block that is not completely edited, and therefore not assembled is transferred to cassette. Provided the EDIT key is depressed beforehand (cold start entry to the editor), all the program data from BEGAD onwards (like ENDAD, BEGAD will have been entered prior to EDIT) up to the address indicated by the current end address pointer CEND, will be copied onto tape in the form of a single data block during the DUMP routine. For such a data block to be transferred by means of the SEF key, it is absolutely imperative BEGAD and CEND are defined, which of course in only possible once the program has been edited! Thus, before SEF is operated to load an edited program onto cassette, the editor must be left by way of the monitor program and then the TM program is started (during this time, the Junior Computer must not be switched off – BEGAD and CEND are stored in RAM on page ØØ!!). What is the procedure?

- Start the TM program via the original monitor program:
  RST Ø 8 1 Ø GO.

After the GO key is depressed, the TM program will be running. Situation (1) in figure 10 will then appear on the display. Now enter any ID between Ø1 . . . FE. This is the only thing that has to be done beforehand, because SA and EA are automatically established by BEGAD and CEND, respectively.

- Introduce an empty section of tape. Be careful not to erase any data already stored on cassette.
- Jot down the value shown in the tape counter and the ID; **SA and EA are not noted until the end of transmission!** This is because the contents of BEGADH, BEGADL, CENDH and CENDL are not transferred to SAH, SAL, EAH and EAL, respectively, until **after** SEF has been depressed.
- Place the recorder in the record mode. Make sure the record level is sufficient and start the tape.
- Operate the SEF key. The data is then stored on tape during the DUMP routine. The display remains unlit after SEF is operated, but the red LED will light. Once all the data has been transferred the first instruction of the edited program that has just been taped will appear on the display. This is because straight after the transmission, the contents of the current address pointer, CURAD, are made equal to those of BEGAD and so the computer makes a warm start entry to the editor.
- Stop the cassette recorder.
- Note the start address (SA) and the end address (EA) of the edited data block. The start address is stored in BEGAD (= ØØE2 and ØØE3) and the end address is stored in CEND (= ØØE8 and ØØE9). This information is only necessary if a warm start entry to the editor is to be made when a file is read back – instead of assembling the program immediately.

The required information can be obtained as follows:

RST	back to the monitor program
Ø 8 1 Ø GO	ID
PAR	(SAH)
PAR	(SAL)
PAR .	(EAH)
PAR	(EAL)
PAR	(BEGH); see SAH
PAR	(BEGL); see SAL
PAR	(ENDH); note if necessary
PAR	(ENDL); note if necessary

Note that the contents of ENDAD have to be at least six more than the contents of CEND. There will be no problems if files in the address range Ø2ØØ . . . Ø7FF are edited. It is advisable therefore, to always set the contents of BEGAD and ENDAD to Ø2ØØ and Ø7FF, respectively.

A warm start entry into the editor can then be made:
RST 1 C C A GO

The BEGAD and ENDAD pointers do not need to be set. The current end address pointer, CEND, will be pointing to the first free memory location after the EOF character 77 and the current address pointer, CURAD, contains the same as BEGAD. The first instruction of the edited block will then be displayed. Again, do not switch off the computer in the meantime!

Why is it necessary to note the start address and the end address? This can be explained in connection with an important feature of the SEF function. The SEF function allows the operator to store incomplete (un-assembled) programs of any length on tape. Supposing, for instance, that you are busy developing a program, which is stored in RAM, by means of the editor. After a certain time you think that you have reached the stage at which the program can be considered complete and ready for testing after its final assembly. More often than not, however, such programs are not ready at all. All sorts of things need modifying. But the labels, of course, have disappeared during the assembly, so that the program as it stands cannot be re-edited. There is nothing for it but to start from scratch! All this can be avoided by storing the original edited version of the program on tape before it is actually assembled. How do we go about this?

- Operate the SEF key and write down the identifier (ID), the start address (SA) and the end address (EA).
- Now start the assembler in the normal manner:
  RST 1 F 5 1 GO
- Start the user program:
  AD (SA) GO
- If there is something wrong with the program, the listing will have to be checked thoroughly. It may well take quite a while to pick out the error, so the computer may just as well be switched off for a while to save energy. After all, everything has been stored on cassette.
- Switch the computer on and carry out the procedure outlined in the section describing the GET function.

Now depress:

```
RST  Ø Ø E 2
DA    pp        BEGADL = pp = noted SAL
+     qq        BEGADH = qq = noted SAH
+     rr        ENDADL   rr = noted ENDADL
+     ss        ENDADH = ss = noted ENDADH
+     pp        CURADL = BEGADL        do not have to
+     qq        CURADH = BEGADH        be entered
+     tt        CENDL    = tt = noted EAL
+     uu        CENDH    = uu = noted EAH
AD 1 C C A GO   warm start entry to editor
```

Since the contents of CURAD are the same as those of BEGAD, the first instruction (or label) will appear on the display after the warm start entry into the editor. Correct the program where necessary and repeat the above procedure as often as required.

- If the program was correct, the edited version will no longer be necessary unless it is to be further developed later on. At least the correct version can now be stored on cassette after assembly. In any case, it is always a good idea to make a complete listing of the finished program. This can be done either as a hex dump (see chapter 12) or as a full assembled listing including comments etc. If the latter method is chosen, a program can then be checked instruction by instruction quite easily by using the SKIP function. Although this was dealt with in chapter 5 of Book 2, a brief recap would not be amiss:
- Set the contents of CURAD equal to those of BEGAD, the start address of the program.
- Set the contents of CEND equal to those of ENDAD. ENDAD may have moved if any corrections have been carried out. Be sure that CEND = ENDAD is pointing to an address which is at least two locations higher up than the last instruction in the program. If this is not done the SKIP function will not operate correctly (EEEEEE will appear on the displays).

If the program requires altering at this stage, SKIP through the program until the EOF character (77) appears on the display. Depress the SKIP key again. Depress the RST key and note the address stored in locations ØØE6 and ØØE7 (CURAD). Enter this address in locations ØØE8 and ØØE9 (CEND) respectively.

- Make a warm start entry into the editor. **Warning.** Never depress EDIT and use a cold start entry into the editor! A cold start is only required if the editor has to start work from scratch. A cold start entry into a data block that is already edited would cause several memory locations to be overwritten by the EOF character. In the case of a warm start entry, the EOF character will be situated at the very end of the file and so no new EOF character will be shifted down the program.

Another warning. If ID = FF and several edited programs are being stored on tape (with the EOF character at the end of each block) the contents of CEND will be the end address of the last file to be read. The contents of ENDAD will therefore have to be readjusted. An edited and taped block (or blocks) may be re-adressed by re-entering it (them) with FF as the ID. It goes without saying that in that case the CURAD and CEND parameters have to be adjusted accordingly.

The PLL control

Reading data without errors

When data is read from cassette the PLL constitutes the ears of the Junior Computer. It is therefore extremely important to ensure that the PLL receives the correct audio signals.

The PLL is calibrated with the aid of the multi-turn preset potentiometer, P1, on the interface board (see figure 2 in chapter 10). The signal received by the PLL consists of a series of alternating 3600 Hz and 2400 Hz tones. In the absence of an input signal the free-running VCO frequency is determined by the setting of P1. This should be halfway between 3600 Hz and 2400 Hz, that is to say around 3000 Hz. If the VCO frequency of the PLL is greater than 3600 Hz or less than 2400 Hz, the PLL will not 'lock' and the output signal will be permanently high or low. It is important, therefore, to set P1 in its mid position before calibrating the PLL.

There are two methods of calibrating the PLL:

- Adjust P1 and keep an eye on the display while the cassette recorder sends out previously recorded test data. Preset P1 is initially set in the mid position and is rotated to the left and to the right until a happy medium is obtained. This is known as the empirical method . . .
- A more scientific approach is to monitor the output signal from the PLL on an oscilloscope and adjust P1 during the test data transmission until the signal is up to standard.

The ratios of 2:1 and 1:2 enable a clear enough distinction to be made between the logic levels of the data bits transmitted by the cassette. Nevertheless, the PLL still needs to be adjusted with due care. This is what has to be done:

1. Calibrate the PLL with the aid of the display

Obviously, IC4 will have to be an EPROM containing the TM program. We realise that this has been mentioned a number of times, but the point has to be made!

Two short routines (see table 2) co-operate in the setting up procedure. The first is located in the address range Ø2ØØ . . . Ø25Ø and uses one TM subroutine to provide about 4 minutes of synchronisation characters which are recorded on tape. The second is located in the address range Ø251 . . . Ø283 and monitors the played back synchronisation characters with the assistance of four TM subroutines.

P1 should be adjusted in such a way that these characters are read from tape correctly. This can be seen on the display: the configurations that are likely to appear are shown in figure 7. The preset potentiometer will be correctly adjusted when the situation shown in the second drawing of figure 7 is stable. That is to say, when the display does not flicker between situations (1) and (2).

The calibration procedure takes place as follows:

a. The machine is switched on and the programs given in table 2 are entered.

Table 2. Two routines which enable test data (synchronisation characters) to be stored on and read from tape, respectively, and which allow the PLL to be calibrated according to the 'display method'.

```
M
HEXDUMP:  200,250
         Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
Ø2ØØ:  A9 7D 8D 6C 1A A9 C3 8D 6D 1A A9 Ø3 8D 76 1A A9
Ø21Ø:  Ø2 8D 77 1A A9 47 A2 FF 8D 82 1A 8D 78 1A 8E 83
Ø22Ø:  1A A9 ØØ A2 7F 8D 8Ø 1A 8E 81 1A A9 DD 8D ØØ 1A
Ø23Ø:  8D Ø1 1A 18 A9 Ø1 6D ØØ 1A 8D ØØ 1A A9 ØØ 6D Ø1
Ø24Ø:  1A 8D Ø1 1A BØ Ø8 A9 16 2Ø A3 ØA 4C 33 Ø2 4C 1D
Ø25Ø:  1C
JUNIOR
M
HEXDUMP:  251,283
         Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
Ø251:  A9 32 8D 82 1A 8D 78 1A A9 7E 8D 83 1A A9 7F 8D
Ø261:  81 1A A9 FF 8D 6B 1A 2Ø C2 ØB 6E 6B 1A AD 6B 1A
Ø271:  2Ø E8 ØB C9 16 DØ FØ 2Ø 36 ØC 2Ø 5D ØC C9 16 FØ
Ø281:  F6 DØ DF
JUNIOR
```

b. The cassette recorder is connected to the computer. Preset potentiometer P2 is turned fully clockwise and the record level control of the cassette recorder is set in the mid position.

c. The recorder is set to 'record' and is started. Enter AD Ø2ØØ GO. The red LED lights and the synchronisation characters are recorded.

d. After about four minutes the write operation will have been completed. The red LED will go out and Ø2ØØ A9 will appear on the display. The cassette recorder is then stopped and the tape rewound.

e. Set the recorder to 'play' (read) this time and enter: AD Ø251 GO. If the headphone socket of the cassette recorder is used, turn the volume control up half-way. The green LED lights. During the section of tape preceding the synchronisation characters, situation (1) in figure 7 will appear on the display. This will flash! However, as soon as the synchronisation characters are being read, the PLL can be calibrated.

f. Adjust P1 until the second drawing in figure 7 appears on the display. If this does not alter while the remainder of the synchronisation characters are being read, P1 will have been set correctly.

To be absolutely sure this procedure should be repeated a few times. We are not going to tell you in which direction to turn P1, as this depends on certain parameters of the PLL circuit which are rather complicated and can not be dealt with here.

That covers the first method of setting up the PLL. Now, how about the second?

2. Calibrate the PLL using an oscilloscope

Readers who are lucky enough to own an oscilloscope (or possibly borrow one!) can calibrate the PLL quite easily. A dual-trace oscilloscope is ideal, but not strictly necessary. To start with, a section of memory is recorded on cassette with the aid of the routine given in table 3. This can be of any length.

The data is then replayed. The output signal from the tape recorder will be presented to the input of the PLL, which is pin 7 of IC7 (the junction between resistors R30 and R31) in the diagram shown in figure 2 of chapter 10. This point is monitored by the oscilloscope. Preset potentiometer P1 will be correctly adjusted when the drawing in figure 12c and

Table 3. This routine transfers test data to the cassette recorder (until the RST key is depressed) and allows the PLL to be calibrated with the aid of an oscilloscope.

```
M
HEXDUMP:  200,23F
         Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
Ø2ØØ:  A9 7D 8D 6C 1A A9 C3 8D 6D 1A A9 Ø3 8D 76 1A A9
Ø21Ø:  Ø2 8D 77 1A A9 47 A2 FF 8D 82 1A 8D 78 1A 8E 83
Ø22Ø:  1A A9 ØØ A2 7F 8D 8Ø 1A 8E 81 1A 2Ø C8 ØA 2Ø E5
Ø23Ø:  ØA 2Ø E5 ØA 2Ø C8 ØA 2Ø C8 ØA 2Ø E5 ØA 4C 2B Ø2
Ø24Ø:
JUNIOR
```

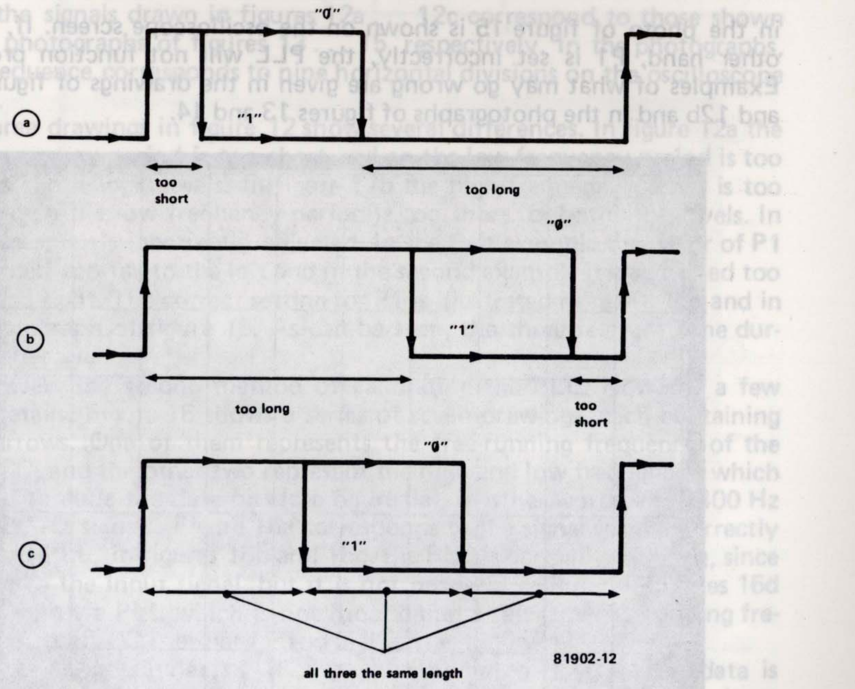


Figure 12. A series of three (possible) output signals from the PLL. Figures 12a and 12b correspond to an incorrectly calibrated PLL and figure 12c shows the result of a correctly calibrated PLL.

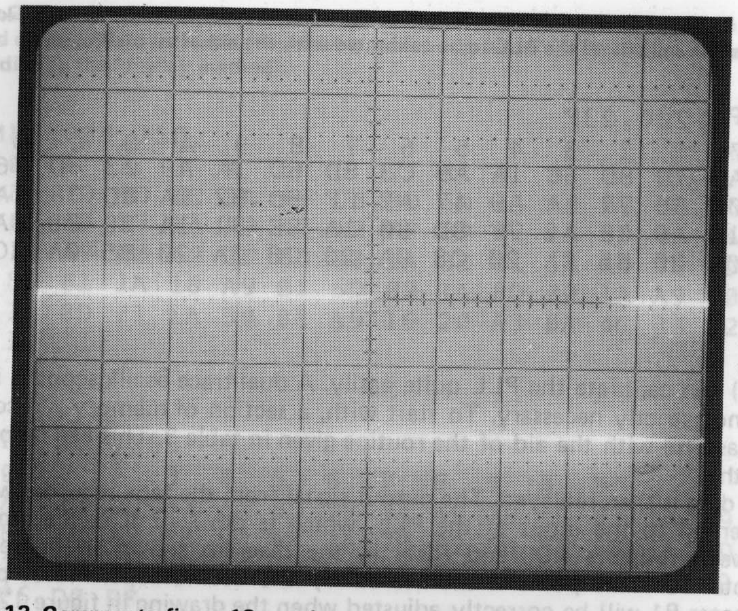


Figure 13. Compare to figure 12a: an incorrectly calibrated PLL.

in the photo of figure 15 is shown on the oscilloscope screen. If, on the other hand, P1 is set incorrectly, the PLL will not function properly. Examples of what may go wrong are given in the drawings of figures 12a and 12b and in the photographs of figures 13 and 14.

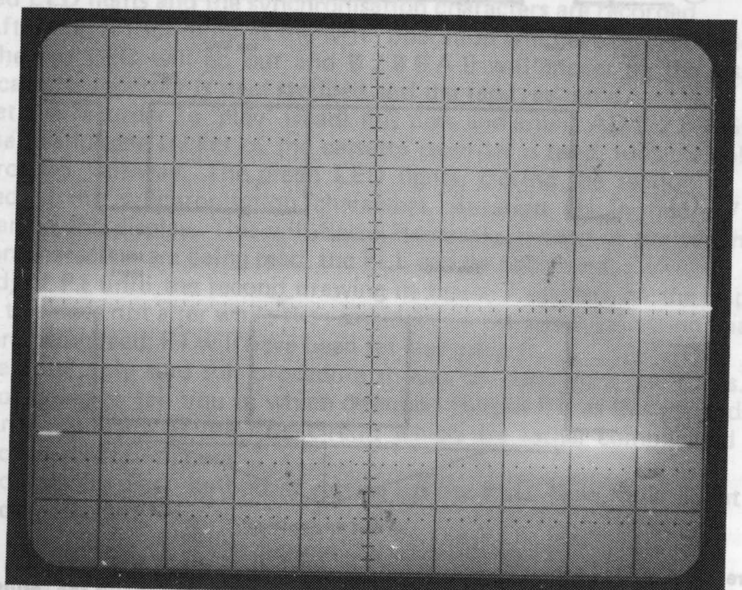


Figure 14. Compare to figure 12b: an incorrectly calibrated PLL.

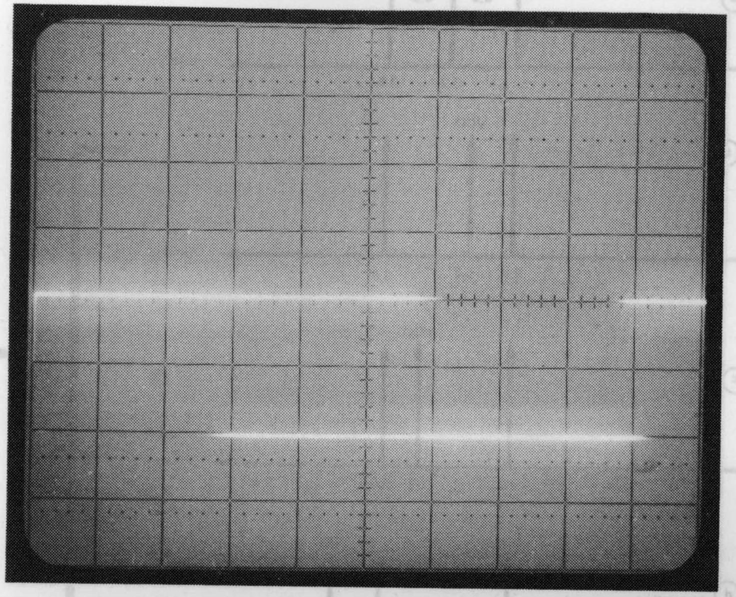


Figure 15. Compare to figure 12c: a correctly calibrated PLL.

Thus the signals drawn in figures 12a . . . 12c correspond to those shown in the photographs of figures 13 . . . 15, respectively. In the photographs, a bit sequence corresponds to nine horizontal divisions on the oscilloscope screen.

The three drawings in figure 12 show several differences. In figure 12a the high frequency period is too short and so the low frequency period is too long for both logic levels. In figure 12b the high frequency period is too long and so the low frequency period is too short for both logic levels. In both cases, P1 is incorrectly adjusted. In the first example the wiper of P1 was turned too far to the left and in the second example it was turned too far to the right. The correct setting for P1 is illustrated in figure 12c and in the photograph of figure 15. As can be seen, the three relevant time durations are equal.

That covers the second method of calibrating the PLL. Now for a few more details. Figure 16 shows a series of seven drawings, each containing three arrows. One of them represents the free-running frequency of the PLL VCO, and the other two represent the high and low frequencies which are used to code the data bits (see figure 6a), in other words, the 2400 Hz and 3600 Hz signals. Figure 16a corresponds to the signal from a correctly calibrated PLL. In figures 16b and 16c, the PLL is certainly working, since it reacts to the input signal, but it is not properly calibrated. Figures 16d and 16e show a PLL which is not functioning at all (the free-running frequency of the VCO is either far too high or far too low).

Figure 16f looks similar to 16b: it shows what happens when data is recorded on one machine and played back on another, the latter having a tape speed that is 10% faster than the former. The nominal speed is of course 4.76 cm per second, but this may often vary by ± 10%. Figure 16g

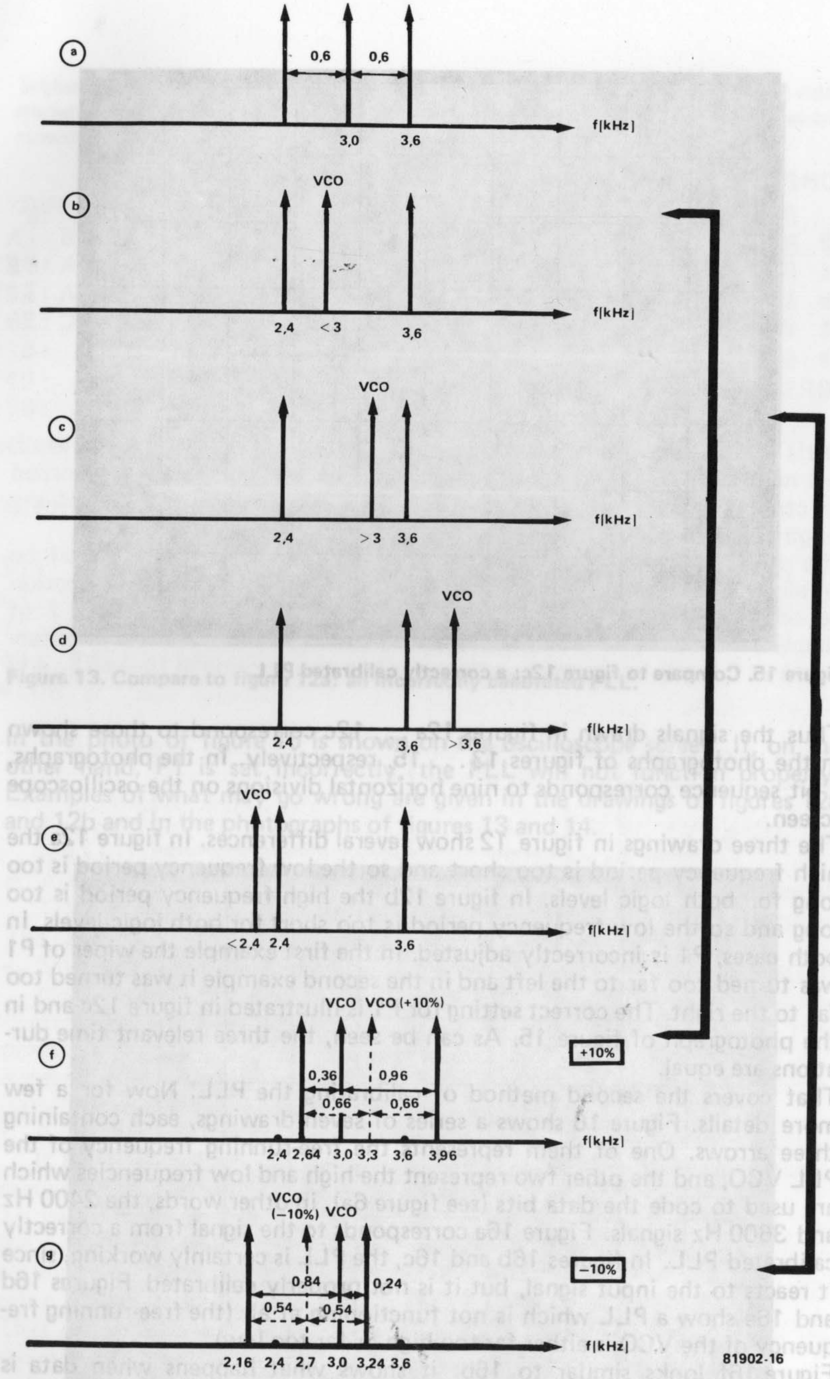


Figure 16. The seven drawings show three different frequencies: the high (3600 Hz) and low (2400 Hz) frequency data signals and the centre (free-running) frequency of the PLL VCO. The latter is preset with P1 on the interface board.

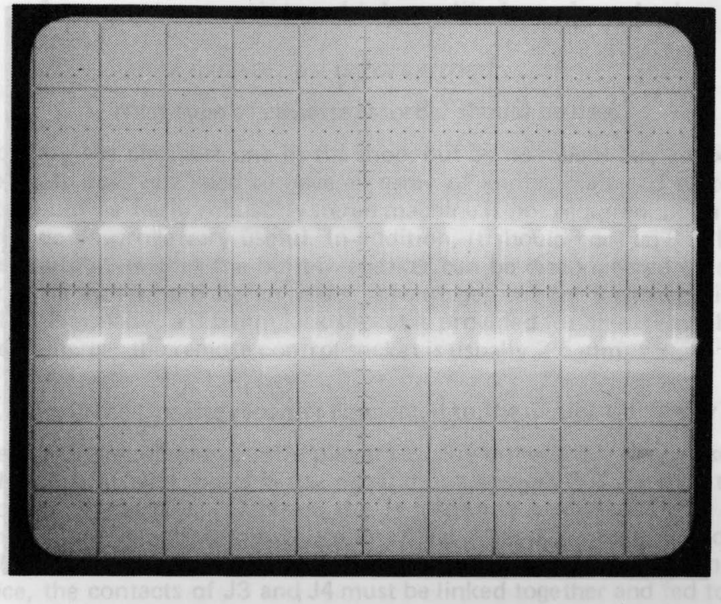


Figure 17. The output signal at the wiper of P2 during a data transmission to tape using the routine given in table 3 (2 V/cm).

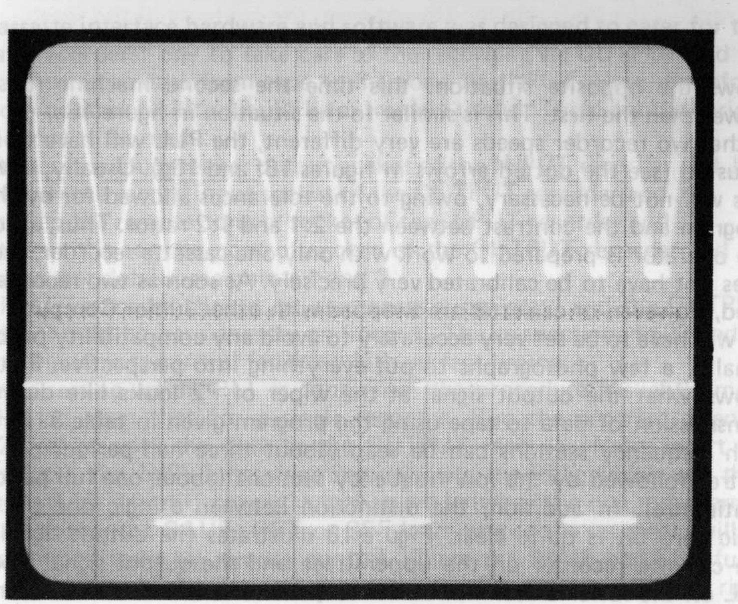


Figure 18. The output of the tape recorder (upper trace) and the output of the PLL (lower trace) during the loading of data from cassette. The volume control of the cassette recorder is at maximum level. (Both traces: 2 V/cm).

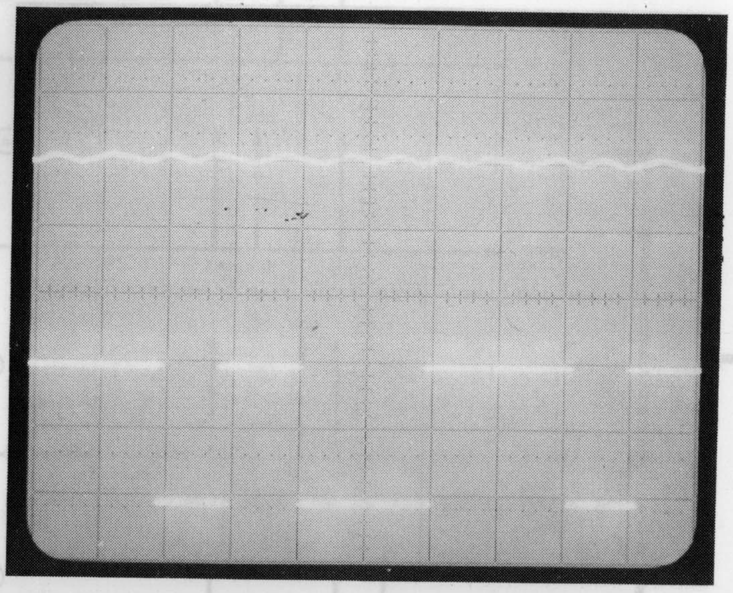


Figure 19. Similar to figure 18, but now the volume control of the cassette recorder has been turned down. The PLL operates satisfactorily over a wide range of input levels. However, higher input levels (figure 18) will give more reliable results. (Both traces: 2 V/cm).

shows the opposite situation: this time the second machine runs 10% slower than the first. This is similar to the situation in figure 16c.

If the two recorder speeds are very different, the PLL will have to be re-adjusted (see the dotted arrows in figures 16f and 16g). Usually, however, this will not be necessary, owing to the tolerances allowed for by the TM program and the contrast between the 2:1 and 1:2 ratios. Thus, as long as the operator is prepared to work with only one cassette recorder, the PLL does not have to be calibrated very precisely. As soon as two recorders are used, however, or cassettes are swapped with other Junior Computer users, P1 will have to be set very accurately to avoid any compatibility problems.

Finally, a few photographs to put everything into perspective. Figure 17 shows what the output signal at the wiper of P2 looks like during the transmission of data to tape using the program given in table 3. First the high frequency sections can be seen (about three half-periods per centimetre) followed by the low frequency sections (about one full period per centimetre). In addition, the distinction between a logic one bit and a logic zero bit is quite clear. Figure 18 illustrates the output signal from the cassette recorder on the upper trace and the output signal from the PLL on the lower trace. As can be seen, the cassette recorder converts the symmetrical squarewave signal in figure 17 into a form of sinewave. This is not that important, as the PLL is a frequency, rather than waveform, detector.

Answers to questions which are likely to be asked

Forewarned is forearmed . . .

1. What type of cassette recorder should be used?

Do not buy the cheapest one in the shop, but by no means buy an expensive one. It does not need to have an array of knobs, dials and switches, but it should be fairly robust. A stereo machine is not required, but a tape counter would prove very useful. In addition, it should have an 8 Ω loudspeaker output, so that the built-in speaker can be disconnected. Remote control facilities would be an added advantage, but by no means indispensable. Normally, a 3.5 mm jack socket is provided for an external loudspeaker, whereas the remote control socket is usually 2.5 mm.

2. How is the cassette recorder connected to the Junior Computer?

The record input of the cassette player (usually either a 3.5 mm socket or a DIN socket; if DIN, pin 3 is the signal input and pin 2 is ground) to J2 (the chassis connector). Connect the loudspeaker output (3.5 mm jack socket) to J1. The remote control connections, J3 and J4, are not used.

N.B. Remote control of one tape recorder is possible, if desired. In this instance, the contacts of J3 and J4 must be linked together and fed to the remote control input. Furthermore, the two relays, Re1 and Re2, must be plugged in.

3. How are two cassette recorders connected to the Junior Computer?

The cassette interface hardware and software was designed to cater for two separate recorders: one to take care of the recording via OUTPUT and the other to transfer data from tape to memory via INPUT. Here, therefore, a remote control facility would be extremely useful. These are the various connections to be made:

a. Between the loudspeaker output socket of the INPUT recorder and J1.
b. Between the record input socket of the OUTPUT recorder and J2.
c. Between the remote control socket of the INPUT recorder and J3.
d. Between the remote control socket of the OUTPUT recorder and J4. For further details, see points 1 and 2.

The INPUT recorder should be permanently on 'play' and the OUTPUT recorder should be permanently on 'record'. The connections to J3 and J4 activate the remote control facilities with perfect timing.

As far as setting up the PLL is concerned, the procedure is slightly more complicated than that for a single recorder. Run the program given in table 2 and transfer the data to the OUTPUT recorder. Next, insert the cassette into the INPUT recorder and calibrate the PLL, which will then compensate for any differences in tape speed between the two recorders.

After operating the SAVE, GET or SEF keys, one of the recorders will be started immediately by remote control. Where the SAVE and SEF functions are involved, the cassette recorder should be started at the right speed at once, as otherwise the start of the data transmission will be altered. This is one of the reasons why there are so many synchronisation characters recorded at the start of each transmission (lasting about 2.5 seconds). It does not matter, therefore, if 50 or 100 are lost, if in the meantime the tape speed has settled down.

4. How is the correct recording level determined?

A good recording leads to a reliable data reading. This is, in effect, what sound technicians take years to achieve . . . but do not worry – we are not going into all the vastly complex details here! All we are interested in is finding a suitable recording level, so that the data can be read back without any of it being lost or mutilated on the way. As mentioned earlier, once a signal has been recorded on cassette it has already been mutilated to a certain extent. In other words, it will have to be compensated for during playback. If your cassette recorder has a tone control, it should be turned up to maximum (full treble). Make sure the record and playback heads are clean to prevent any high frequency signals from disappearing.

Up until now we have only concerned ourselves with the external loudspeaker output of the INPUT recorder, because this is where the level really counts. The 'line' output provides a signal level that is too low for the PLL to function correctly.

N.B. **If the line output is used, resistor R37 must be omitted.**

With regard to the actual record level, it is very difficult to give precise instructions. It all depends on the quality of the cassette recorder(s) used. Generally speaking, the record level is unusually high for human ears (fortunately the internal loudspeaker of the recorder is disconnected!). By the way, cassettes do tend to wear out rather quickly and so do record and playback heads. Do not pile up your cassettes near to a mains transformer, as the magnetic field might erase the contents of the tape.

With time you will find out (as we did) that it does no harm to make several recordings of a particular program, preferably on different cassettes (spare copies do come in handy!).

By now, most readers should be able to cope with all the intricacies of the cassette interface, the PLL and the TM program. Nevertheless, it is always better to be safe than sorry and so the remaining pages of this chapter will be devoted to the various things that just might go wrong . . .

5. Does it work?

. . . or how to make a quick diagnosis

By now the hardware extensions should have been built according to the instructions given in chapter 10. Full attention can, therefore, be given to the software.

Is the TM program doing its job correctly?

First and foremost, IC4 should be a 2716 EPROM containing the TM program (never mind about IC5 for the time being!). Look at all the pins and make sure that none of them are bent and that the IC is positioned correctly.

Now let us enter the following sequence:

AD Ø 8 1 Ø GO (note the start address is Ø81Ø and not Ø8ØØ!). The next item to be displayed is: 'id ØØ'. If the EDIT key is operated, 77 should appear on the display (provided the contents of ENDAD are greater than those of BEGAD). If, on the other hand, the SAVE key is depressed, the red LED (OUTPUT) will light and the contacts of J4 will be shorted (check with an ohmmeter) provided Re2 is mounted on the interface board. Data may be transferred from the memory of the Junior Computer even if there is no cassette recorder at the receiving end. Make the contents of EA greater than those of SA and operate the SAVE key. Shortly afterwards, the Junior Computer should report back by displaying 'id xx' (meanwhile, the red LED will light). The SEF function can be tested in the same manner. Enter an imaginary program anywhere in RAM with the aid of the editor. Stick to the rules regarding BEGAD and ENDAD and then operate the SEF. key. Again, the Junior Computer will shortly give a sign of life by displaying 'id xx' and again the red LED will have been lit. The contacts of J4 will, of course, now be open circuit.

It all sounds very good, but what if the above does not happen? The first thing to do is to check the contents of the EPROM, IC4, with the aid of the hex dump given in Appendix 4. Examine the circuit around T3. What about the links between the interface board and the main board port connector? There are five of them . . .

Not only can data be written into thin air, as we did earlier, but data can also be read without a cassette recorder being connected to the Junior Computer. Operate the GET key and wait for the green (INPUT) LED to light. If relay Re1 is where it should be, the contacts of J3 will be shorted. After the GET key is depressed, the first situation in figure 7 will be displayed. The display will probably be a little faint and will be seen to flicker quite clearly.

If everything has gone well up to now, the PLL can be calibrated. Connect the recorder(s) as indicated earlier. Depending on which calibration method has been chosen, either the two auxiliary routines given in table 2 or the write routine given in table 3 may be used. Then the actual calibration procedure may be carried out (see the relevant paragraphs earlier in this chapter).

If the calibration is not satisfactory, check the following points:

- Were the test routines entered without any errors?
- Is P2 turned fully clockwise? Its position has a direct influence on the signal sent to the OUTPUT recorder.
- Examine the connections between the interface and the cassette recorder(s).
- Is P1 set to the mid position? If not, the PLL will not work (see figures 16d and 16e and the corresponding text).
- Are the record and playback levels sufficiently high?
- Look at the circuit around the PLL; do IC6 and IC7 plus associated components meet all the requirements? Check all the supply voltages (see figure 18 in chapter 10).
- Is resistor R37 still on the board even though you are using the line output? If so, remove it!
- Readers with access to an oscilloscope should compare the various signals with those shown in figures 13 . . . 15 and 17 . . . 19. No problem is impossible to solve! Usually you will find human inaccuracy is at the bottom of it! (We know from experience!). If you are really stuck, contact

the technical editorial staff at Elektor either by letter or by telephone on Monday afternoons.

Final check

Once everything is in proper working order and the PLL has been calibrated, a test data block can be entered to see whether or not it is transmitted and received correctly. What happens is that a program is stored on cassette byte by byte and the Junior Computer is then switched off. The program is then re-entered, after switching the machine back on again, and stored (and checked) in RAM. If the re-entered program works first time, the transmission of data obviously functions correctly. The Junior Computer was switched off so that the entire RAM contents would be totally random and the test program previously entered completely erased.

Is something wrong with the read function?

Quite a bit was said earlier about the control bytes CHKH and CHKL (see figures 3 and 4). The question now is: what happens if the values of CHKL and/or CHKH do not coincide with the values stored on cassette after the data has been re-entered. The latter values were noted (or should have been!) during the recording. Incidentally, only data blocks which have been searched for and located are actually checked.

Supposing the GET key was depressed. The cassette is now played back. When the required file number is found, situation (2) in figure 7 will be displayed first (synchronisation characters) followed by situation (3). The data block is then stored in memory. As soon as the entire file has been entered and both CHKH and CHKL are correct, the TM program will report 'id xx', where xx is either the program number or ØØ or FF. The Junior Computer then leaves the RDTAPE subroutine and jumps back to the main TM routine.

If either CHKH or CHKL do not tally, the machine will remain in the RDTAPE subroutine and so no 'id xx' will be displayed. The display will in fact switch from situation (3) in figure 7 to situation (1) (clean tape) and after a while situation (2) will reappear: the synchronisation characters of the next data file. (If ID = FF this will be loaded into memory). The fact that the display changed from the third situation to the first instead of to 'id xx', means that something went wrong during the data entry.

Should the above occur, the data block can be read in once again (rewind the cassette, the GET key need not be depressed again). If this does not help matters, the error might be due to one of the following:

a. The record/playback levels are too low.
b. The levels are all right, but there are too many 'drop-outs' on the tape. This situation is very unlikely, but you never know!
c. The data block was recorded on a machine with a tape speed that differs greatly from the playback recorder. P1 is probably correctly adjusted, but will have to be altered to cope with the difference in speed.

The situation outlined in c could well occur when one Junior Computer operator exchanges programs on cassette with another. In other words, the programs were taped on one owner's machine and played back on a completely different model. The remedy is to provide every cassette with a section of test data to help calibrate the PLL. This may, for instance, include the first half of the program given in table 2 (4 minutes of synchronisation characters). After entering the read program (second half of table 2), the PLL can be adjusted to cope with any difference in tape speed.

That brings us to the end of chapter 11. Have fun recording (and playing back) programs!

12

Adding peripherals to the Junior Computer

Extending the input and output facilities

The basic Junior Computer keyboard allows a limited number of different functions. These amply serve the single board computer, but tend to fall short when more extensions are added. The same is true of the seven segment displays — they seem unable to present a complete picture. The answer to these shortcomings is to connect certain peripheral equipment, as a result of which the basic operating panel is transformed drastically.

The peripheral equipment we have in mind includes the ASCII keyboard and Elekterminal (mentioned on many occasions during the course of this book). This combination is connected directly to the Junior-Computer and to a normal domestic television set. The television screen offers a lot more scope than a single line of six characters and gives a much clearer indication of what is actually going on. In addition, various comments can be typed in alongside the program instructions, a thing that was totally out of the question previously. This is because the ASCII keyboard provides a full scale repertory of key functions.

The new facilities can be used to full advantage with the aid of the PRINTER MONITOR system program. This is effectively an extended version of the basic monitor program. The PRINTER MONITOR program utilises ten key commands and, depending on the particular function key which was depressed, will report back to the operator via an appropriate message displayed on the TV screen.

All in all, the Junior Computer 'news' can now be 'broadcast' on TV to keep the operator fully up to date on the machine's progress. The 'news reader' in this chapter happens to be the PRINTER MONITOR program, but in the future the operator may well communicate with the Junior Computer in a higher level language such as BASIC.

The addition of a full ASCII keyboard and a video interface constitutes a very important step along the road to maturity. Initially, of course, the original keyboard and display were ideal, as their simple form enabled the apprentice programmer to get to grips with the microcomputer in a very short space of time. In the previous chapter, however, the original keyboard acquired so many new functions that it proved to be quite difficult to distribute them easily among the keys. All five function keys now serve a number of purposes: the original monitor program, the editor and, after its introduction in chapter 11, the TAPE MONITOR program.

Technically speaking, there is always room for a few more functions. But in practice this leads to considerable problems. For one thing, how do you fit all the new names onto the keys? They could well end up being totally illegible! In any case, a lot more than five command keys are required for a fully grown system. The alphabet alone has 26 letters and then what about upper and lower case letters and numerals etc.?

It is the consideration of using high level languages such as BASIC which really decides the matter. After all, ‚GOSUB' looks a lot better, and is a lot easier to understand, than 'PC + FBA'. So, why not make life a lot easier for ourselves and connect an ASCII keyboard, which is all geared to total communication and quite compatible with the Junior Computer and the video interface.

On the original display there is only room enough for a single address and its contents, or for a single 3 byte instruction, or for six data 'nibbles' belonging to a particular user program. Clearly, the time has come for:

1. **A larger selection of displayed characters.** The possibilities for a seven-segment display are rather limited, as not a great many different characters can be shown. We would like to be able to use both upper and lower case letters, numerals, punctuation marks, etc.
2. **Longer lines.** Try as they may, six seven-segment displays can not hold too much information. Imagine the Elekterminal as 64 times seven-segments and you will see what we mean. . .
3. **More lines.** Examining a single line of data at a time is rather like looking at the entire memory contents of the computer through a keyhole! The exercise in both instances is very tiring for the eyes. The Elekterminal, on the other hand, is capable of displaying up to 16 lines at a time. If, instead of a video screen, Junior Computer operators wish to obtain 'hard copies' of their programs etc., they can always add a printer in which case the sky is the limit as far as the number of lines is concerned.

In order to meet these parameters that we have just set ourselves, the

ASCII keyboard and the Elekterminal will have to be connected to the Junior Computer by way of the RS 232 interface. With regard to the video screen, any (black and white) television set will do as long as it is connected via the UHF/VHF modulator. As longstanding readers of Elektor will know, the Elekterminal, ASCII keyboard and UHF/VHF modulator are projects which were published as early as 1978. These now require very minor modifications to be completely compatible with the Junior Computer system.

Since the circuits etc. have been published in Elektor magazine, we will not go into all the specific details again, but will concentrate on the way in which they are linked up to the Junior Computer. Then we will go on to describe the way in which the computer communicates with them and with the outside world. Special attention will be given to the data transmission and reception system (the UART; Universal Asynchronous Receiver/Transmitter). . .

Passing information from the Junior Computer to the peripherals and vice versa

Serial data transmission

The Junior Computer is certainly getting bigger and bigger and is now able to control a number of external devices. It is the RS 232 interface, described in chapter 10, that controls the serial data transfer between the Junior Computer and the peripheral equipment. This is, in fact, bi-directional communication as both the computer and the peripheral devices transmit and receive data. That is why there are two separate lines marked RS 232 in the diagram in figure 1: one represents the flow of data from the Junior Computer and the other represents the flow of data from the peripherals. Each line therefore starts at the output of one device and ends at the input of another.

The serial input of the Elekterminal is indirectly connected to the port line PBØ, which, of course, is programmed as an output. The serial output of the Elekterminal is indirectly connected to port line PA7, which is programmed as an input.

Inside the Elekterminal itself there are far more than two connections. The central control device is the UART, which will be described in greater detail later on. The section in the lower left-hand corner of figure 1 constitutes the ASCII keyboard. This is connected to the UART via an 8 bit data bus. As soon as a key is depressed the corresponding ASCII code appears on the bus. A ninth line (strobe) indicates whether or not any key has been operated. This produces a start signal for the UART to process the parallel data entered from the keyboard.

A seven bit data bus and a single control line connect the UART to the actual video section of the Elekterminal. The ASCII characters transmitted along this (uni-directional) 7 bit data bus are either generated when a key is depressed, or are sent to the video interface from the Junior Computer. In the first instance a further distinction is made between letters, figures and punctuation marks on the one hand and control functions such as Carriage Return (CR), Line Feed (LF) etc. on the other hand. These are mentioned in detail in the actual Elekterminal article. The position of the next character to be written on the screen is indicated by a cursor. This is continually updated as new data is entered by way of the 7 bit data bus.

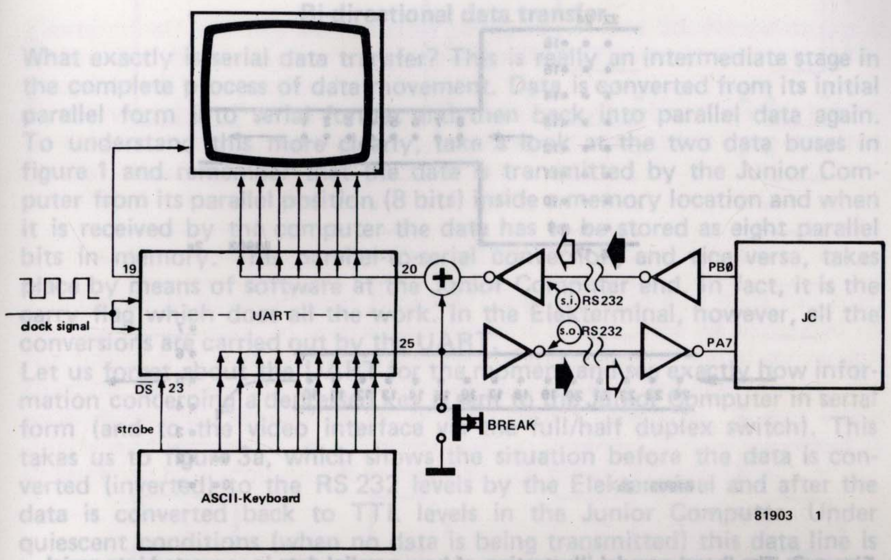


Figure 1. A block diagram showing how data is transferred between the Junior Computer and the peripheral device: the Elekterminal and the ASCII keyboard.

By the way, it is not strictly true that command functions are instigated exclusively by the ASCII keyboard. The computer is also able to output command functions. One or two of the subroutines in the PM program cater for this.

As can be seen from the diagram in figure 1, a clock signal is fed to the UART (extreme left-hand side). This enables the UART to convert serial incoming data into parallel data and vice versa. This may sound complicated, but all that really happens is that data is transferred from a single input line to the wider data bus and vice versa.

The procedure may be compared to a band parade, which at a certain stage in the proceedings has to pass through a narrow alleyway. The members of the band therefore have to start walking in single file. (Whether the bass drummer will get through is, of course, another matter!). Thus the parallel rows of data have to converge to form a single serial file. This procedure is illustrated in figure 2a. The serial-to-parallel conversion is illustrated in figure 2b, where the original condition is restored. Well, not exactly restored, for the configuration is slightly different. Whereas the first band consisted of rows of eight (the data bus between the keyboard and the UART), the second band is made up of rows of seven (the data bus between the UART and the video section).

Figure 1 also shows a 'bridge': the link between the serial output of the UART (the alleyway in figure 2a) and the serial input of the UART by way of a type of summing circuit ('s.i.' and 's.o.' in figure 1 refer to the

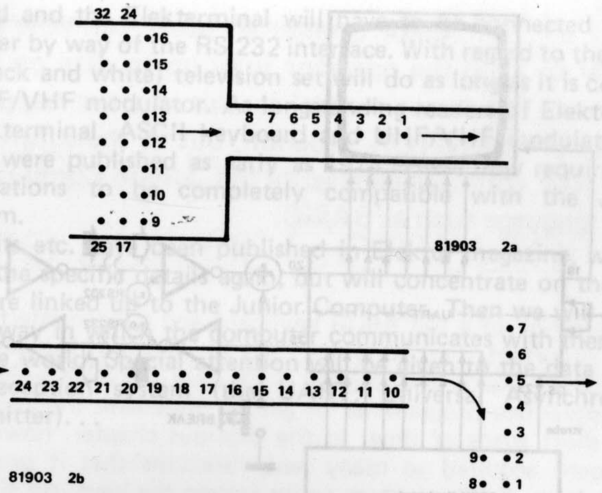


Figure 2. The 'band parade' illustration of how parallel data is converted into serial data (2a), and how serial data is converted into parallel data (2b).

serial input and output, respectively, of the Elekterminal, not those of the UART!). Serial data meant for the video terminal must be derived from the Junior Computer or from the keyboard, but not both at the same time. The link between the input and the output of the UART enables data to be sent to the video interface and to the computer simultaneously. This full/half duplex function is required when there is no need for the computer to confirm the operation of a key. In other words, even when there is no need for the computer to report back to the operator, he/she can keep an eye on the video screen to see what is going on.

Figure 1 also shows a BREAK key. When this is depressed the data line between the Elekterminal and the computer is taken to logic zero. It is used to call the attention of the Junior Computer when, for instance, it is busy sending a particular message. It may not be very polite to interrupt the machine in this manner, but it can be very handy at times.

All that remains to be discussed in figure 1 are the four inverters. These are required to prepare the serial data prior to transmission over the RS 232 lines and to translate the data back to TTL levels during reception. Each data line contains two inverters and readers may wonder about the need for them. After all, if a signal is inverted twice it ends up as if it had not been inverted at all! This is not strictly true in this instance, as the RS 232 standard is based on 'negative logic', where logic one is low and logic zero is high. Both the Junior Computer and the Elekterminal operate on positive logic, where logic one is high and logic zero is low. Therefore, if it were not for the inverters at both ends of the line, the ASCII code involved would be inverted, so that the code for A would not be 41 but BE. As you can imagine, this is extremely impractical so there really is a serious requirement for both inverters.

Bi-directional data transfer

What exactly is serial data transfer? This is really an intermediate stage in the complete process of data movement. Data is converted from its initial parallel form into serial format and then back into parallel data again. To understand this more clearly, take a look at the two data buses in figure 1 and remember that the data is transmitted by the Junior Computer from its parallel position (8 bits) inside a memory location and when it is received by the computer the data has to be stored as eight parallel bits in memory. This parallel-to-serial conversion, and vice versa, takes place by means of software at the Junior Computer end. In fact, it is the carry flag which does all the work. In the Elekterminal, however, all the conversions are carried out by the UART.

Let us forget about the UART for the moment and see exactly how information concerning a depressed key is sent to the Junior Computer in serial form (and to the video interface via the full/half duplex switch). This takes us to figure 3a, which shows the situation before the data is converted (inverted) to the RS 232 levels by the Elekterminal and after the data is converted back to TTL levels in the Junior Computer. Under quiescent conditions (when no data is being transmitted) this data line is held high (logic one). The equipment at the other end of the RS 232 line is waiting patiently for an ASCII character to be transmitted. How does it know when a character is to be sent? Simple. The data line is pulled down (logic zero) for a certain period of time. This duration is called the **bit period** and the change in logic level occurs when the **start bit** is sent.

When the computer detects a start bit (change in logic level), it must wait 1½ bit periods to be absolutely sure that the next logic level it detects belongs to bit Ø. If the computer then waits for a further full bit period, it will be able to detect bit 1, and so on. Bits bØ . . . b7 correspond to the parallel data transmitted by the keyboard. Bit bØ represents the least significant bit in the 7 bit ASCII code and bit b6 is the most significant bit. The eighth bit, b7, is actually the **parity bit.**

The parity bit is generated so that the computer can check whether or not the particular ASCII character has been transmitted correctly. Although this facility is not used in the Junior Computer (see figure 3c), a few words should be said about its purpose. The parity bit can be set or reset before the transmission to bring the number of logic ones in the data word to an even total (= even parity) or to an odd total (= odd parity). During the reception of this ASCII character, the parity bit can be tested to see whether anything has been altered. If so, this is reported and suitable action is taken to rectify the situation. For instance, the transmitting device could well be asked to send the character again. Incidentally, this method is far from perfect!

Once the eight bits have been transmitted, the data line will have to assume its quiescent level (logic one) once more. This is accomplished by transmitting one or two **stop bits.** In our particular instance, two stop bits are transmitted. Figure 3b illustrates the RS 232 line levels corresponding to figure 3a. As can be seen, the inversion has exchanged the high and low logic levels. This is, of course, on the assumption that diode D4 was not mounted on the Elekterminal board (TTL adaptation).

What about data transmission from the Junior Computer to the

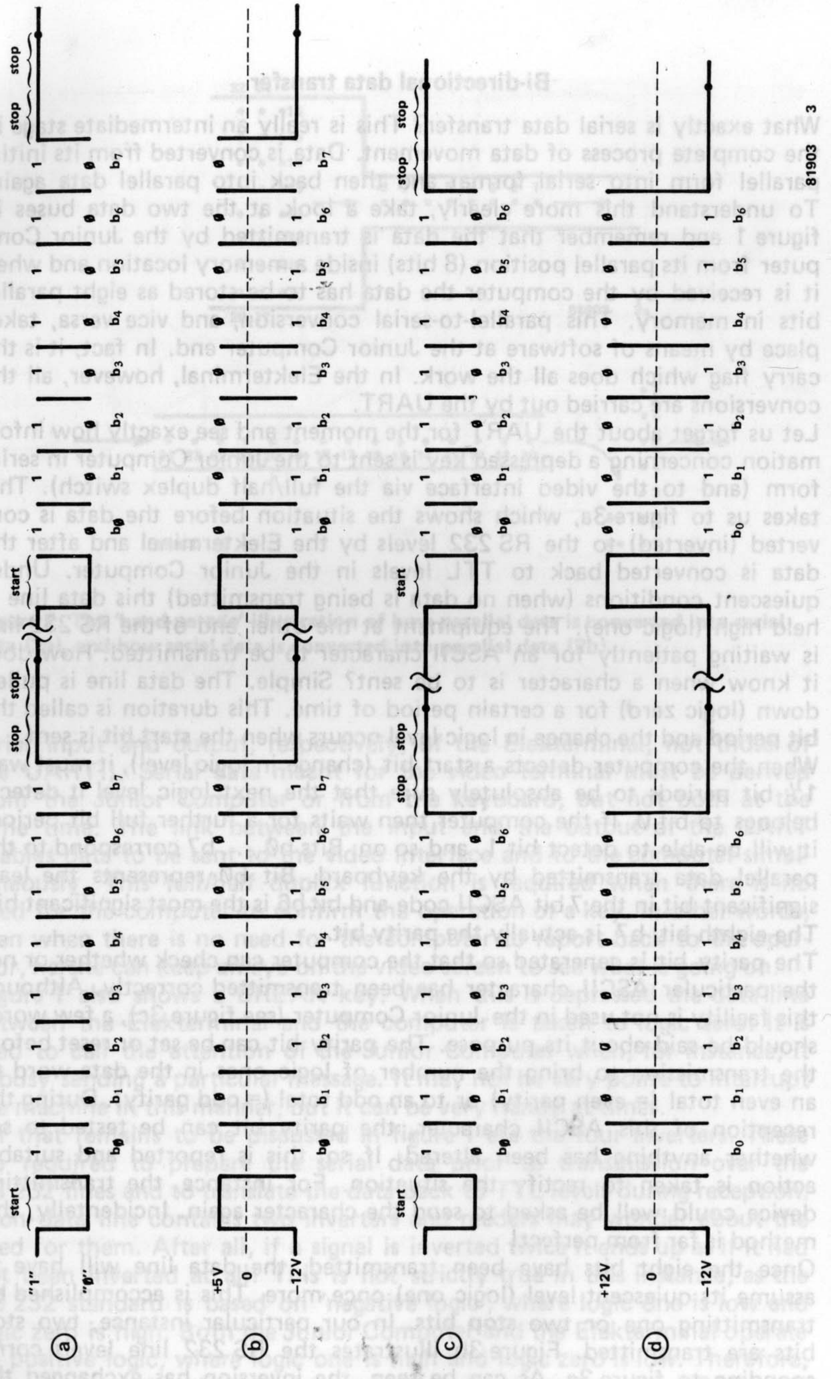


Figure 3. The logic levels (3a and 3c) and the voltage levels (3b and 3d) during the transmission of two ASCII characters from the keyboard to the computer (3a and 3b) via the UART and from the computer to the video terminal via the UART (3c and 3d).

Elekterminal? For this we can refer to figures 3c and 3d. No parity bit is transmitted here. Furthermore, the high RS 232 logic level is at +12 V instead of +5 V as previously.
By the way, bit b7 is in fact transmitted from the keyboard via the UART to the Junior Computer for parity checks, but after it has been received by the computer it is made logic zero.

The UART

The Universal Asynchronous Receiver/Transmitter, to give it its full title, has been mentioned in several places and it is high time we explained its full function. As we know, it takes care of the parallel-to-serial conversion from the keyboard to the Junior Computer and the video interface and the serial-to-parallel conversion from the Junior Computer to the video interface. The system is 'asynchronous' in that the conversion speed is determined by the bit period as opposed to a separate clock signal.

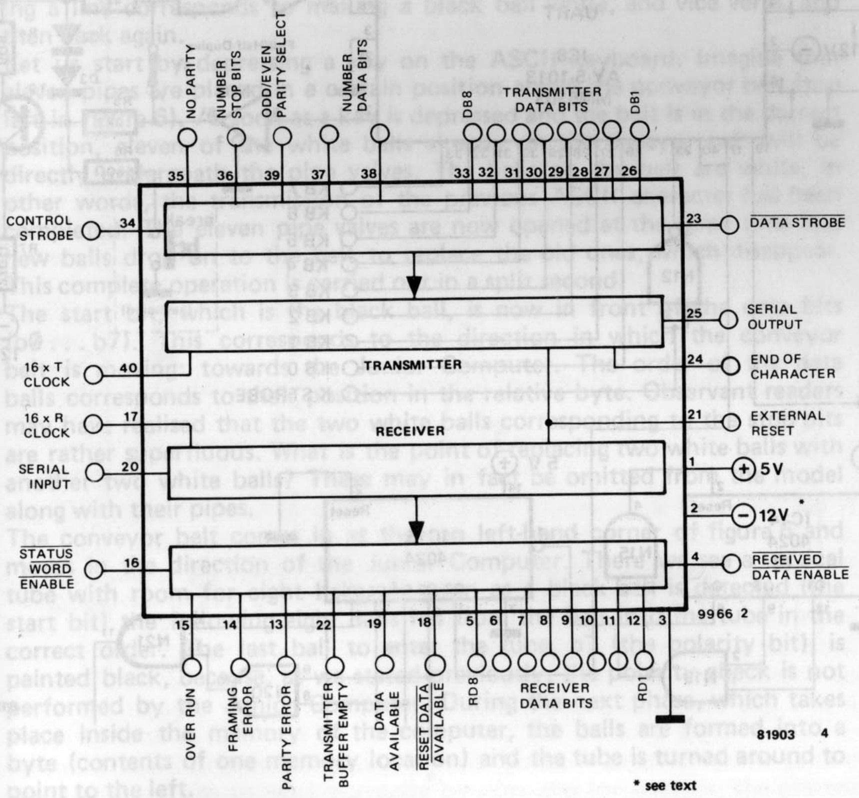


Figure 4. Simplified internal block diagram and pin connections of the UART used in the Elekterminal.

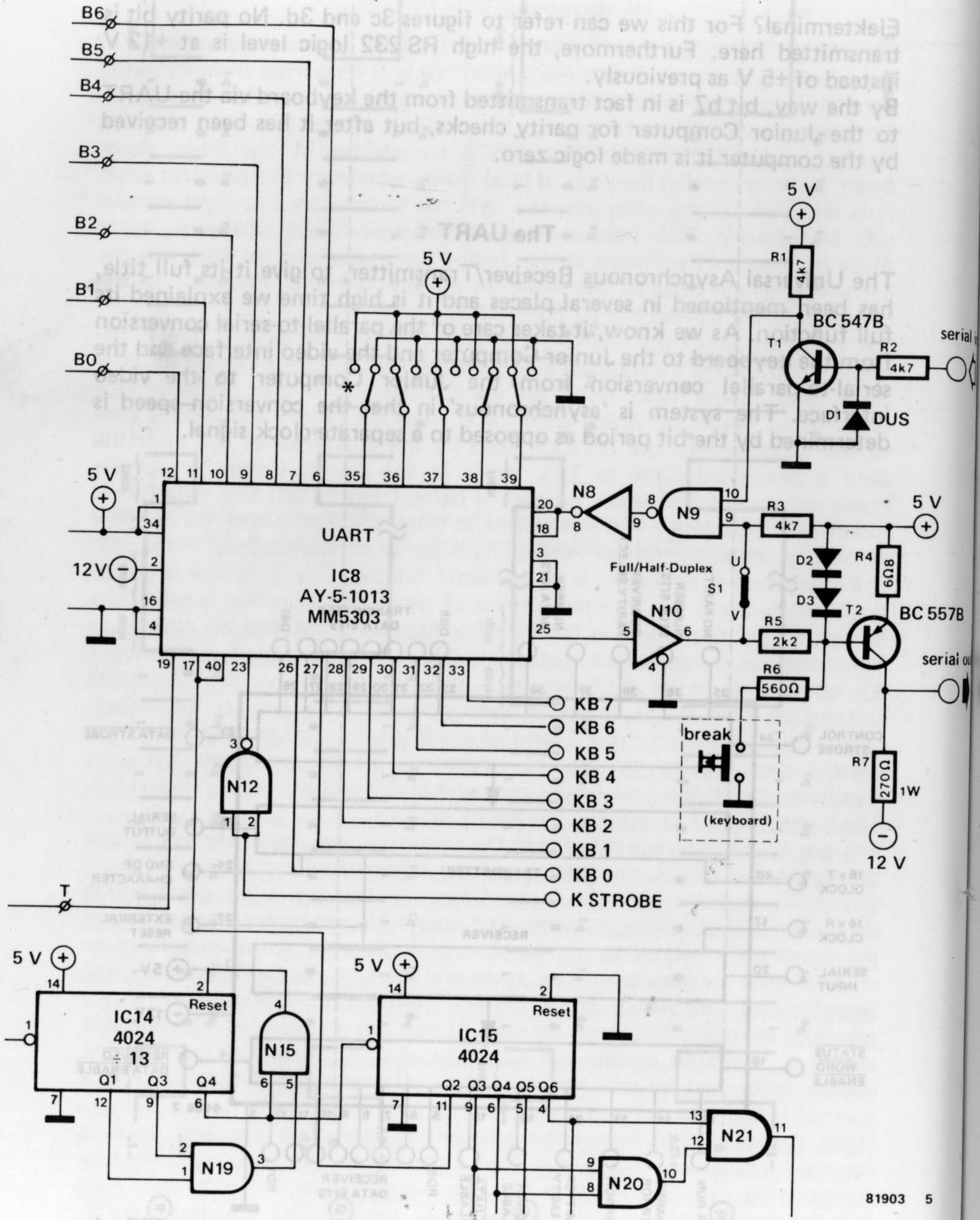


Figure 5. The section of the Elekterminal circuit diagram containing the UART has been slightly modified for use with the Junior Computer.

When the Elekterminal was first published in Elektor magazine (December 1978), the UART was described in detail. Just to recap on this, the simplified internal block diagram of the UART and its pin connections have been drawn in figure 4. Its (electronic) location in the Elekterminal is given in figure 5. The circuit has been modified here and there, but this will be dealt with later during the constructional details. Figure 6 provides an overview of the UART as a pair of conveyer belts. It is this figure which concerns us at the moment.

Consider two conveyor belts: one runs from the Junior Computer to the Elekterminal and the other runs from the Elekterminal to the Junior Computer (the two serial data lines). Each belt carries a number of black and white balls. A white ball represents a logic one and a black ball represents logic zero. A ball marked with a cross in the centre stands for 'don't care'; these may be black or white (the data bits). When the belt has travelled the length of the diameter of a ball, a full bit period will have passed. The speed of the belt therefore increases as the bit rate is reduced. This means that the baud rate — the transmission speed in bits per second — also increases.

Figure 6 shows the situation at both ends of the RS 232 data lines. Crossing a line corresponds to making a black ball white, and vice versa, and then back again.

Let us start by depressing a key on the ASCII keyboard. Imagine that eleven pipes are placed in a certain position above the conveyor belt (top left in figure 6). As soon as a key is depressed and the belt is in the correct position, eleven of the white balls already on the coveyor belt will be directly underneath the pipe valves. The balls on the belt are white, in other words, the transmission of the previous ASCII character has been completed. The eleven pipe valves are now opened at the same time and new balls drop on to the belt to replace the old ones, which disappear. This complete operation is carried out in a split second.

The start bit, which is the black ball, is now in front of the data bits (bØ . . . b7). This corresponds to the direction in which the conveyor belt is moving: towards the Junior Computer. The order of the data balls corresponds to their position in the relative byte. Observant readers may have realised that the two white balls corresponding to the stop bits are rather superfluous. What is the point of replacing two white balls with another two white balls? These may in fact be omitted from the model along with their pipes.

The conveyor belt comes in at the top left-hand corner of figure 6 and moves in the direction of the Junior Computer. There we see a vertical tube with room for eight balls. As soon as a black ball is detected (the start bit) the following eight balls fall from the belt into the tube in the correct order. The last ball to enter the tube, b7 (the polarity bit), is painted black, because, as we stated previously, the polarity check is not performed by the Junior Computer. During the next phase, which takes place inside the memory of the computer, the balls are formed into a byte (contents of one memory location) and the tube is turned around to point to the left.

Now for the lower conveyor belt in figure 6. This runs from the Junior Computer to the Elekterminal. Logically, therefore, the process starts at

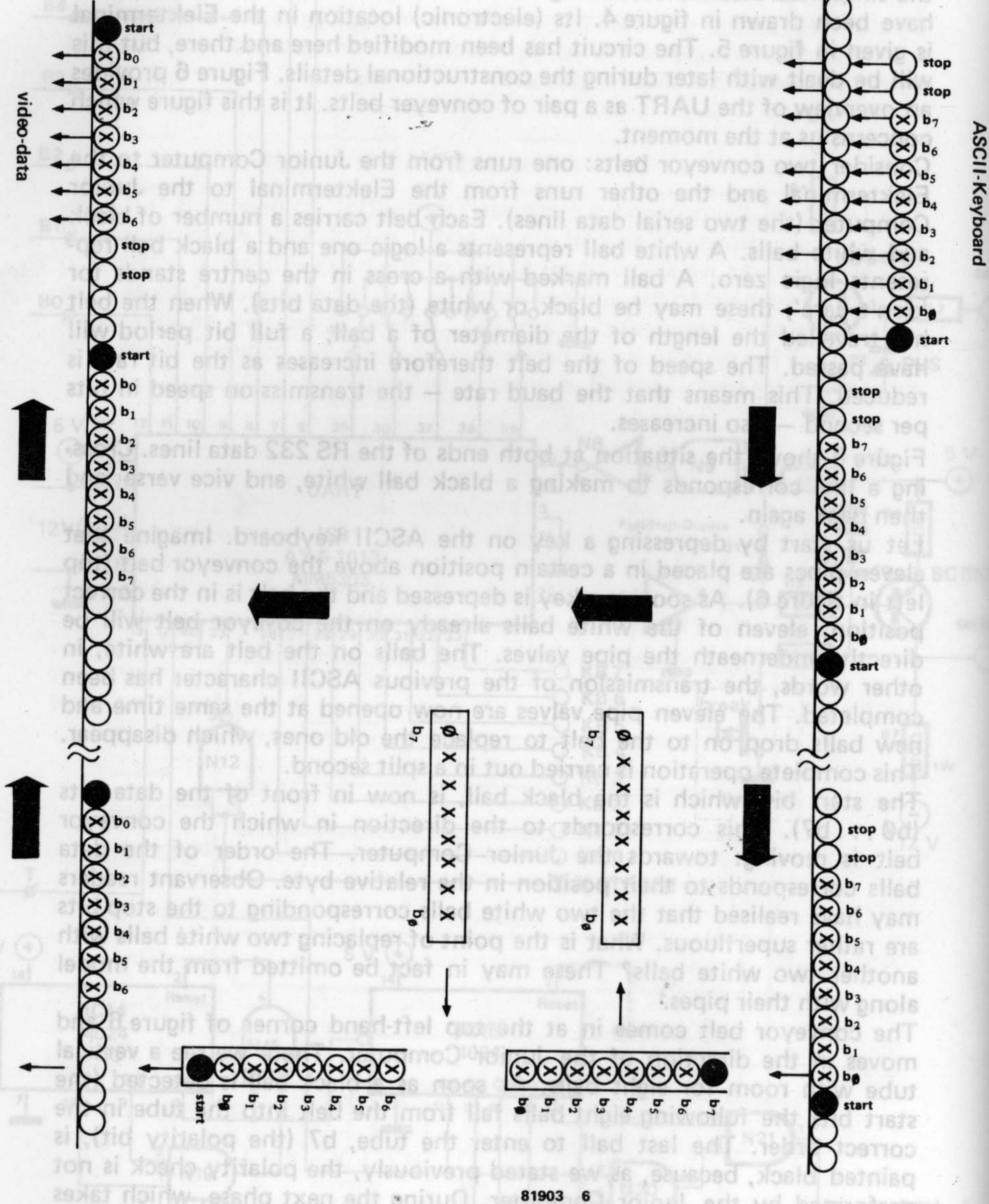


Figure 6. The conveyor belt illustration of how data is converted from parallel-to-serial and from serial-to-parallel.

the computer. The contents of a memory location are transferred into a tube which hangs vertically over the belt and has room enough for eight balls. As soon as there are a series of white balls underneath the tube (to indicate the end of transmission of the previous ASCII character, including the two white stop bits), and one of them arrives directly underneath the pipe valve, the latter opens to let out one ball which pushes the other one off the belt. One bit period later the next ball is pumped out and so on until the tube is empty.

Again, we have the problem of the two white balls which are supposed to replace their predecessors on the belt. In reality the stop bits are transmitted by restoring the quiescent level of the data line (logic high), if necessary, depending on the value of the last data bit to be sent and the transmission of the next ASCII character is disabled for two full bit periods. A similar situation occurs during reception. Immediately after the last data bit has been received the following ASCII character is delayed for two successive bit periods.

Let us get back to the balls on the conveyor belt. As soon as a start bit is detected by the UART in the Elekterminal (the first black ball after at least two white balls), the next seven balls are unloaded from the belt simultaneously. These contain the 7 bit ASCII character data or the control function data pertinent to the video display.

One other thing needs to be said about figure 6. Readers may have noticed the black arrows in the centre of the drawing. These indicate that the serial data stream transmitted from the keyboard not only reaches the Junior Computer, but also the Elekterminal via an internal 'corridor': the hardware echo we mentioned previously. The diagram in figure 6 is a little vague on this point, but users will learn from experience that the data sent from the keyboard travels to the Elekterminal at exactly the same speed as it does to the Junior Computer. Since the computer takes a certain amount of time to react to the data, the serial input of the UART is unlikely to have to process data from two directions at once.

Building the Elekterminal and the ASCII keyboard

How to construct the video interface and connect it to the Junior Computer

As mentioned earlier in this chapter, the Elekterminal and the ASCII keyboard are existing devices which were published in Elektor in 1978. These designs were also featured in the book: 'SC/MPuter (2) build your own microprocessor system', currently available from Elektor. This book and the magazine articles contain full details of the circuit diagrams and the printed circuit boards, as well as explanations as to how they work and constructional hints. Unfortunately, we do not have the room to go into the designs in too much detail here. We can, however, give you a helping hand in the way of construction. This is really quite a straightforward job. For one thing, readers will be quite experienced in getting electronic circuits to work correctly by now and for another, the printed circuit boards involved are single-sided and so should not present any difficulties.

The keyboard

When constructing the ASCII keyboard, the following points should be borne in mind:

- The RES key (optional reset) which was previously not electrically connected to the key matrix (XØ . . . X7/YØ . . . Y1Ø) is now required.

Figure 7 shows that one contact of the keyswitch is connected to row X3 and the other to column Y1Ø in the matrix. This means that the RES switch will now act as 'DELETE' or 'RUBOUT'. When this key is depressed the PRINTER MONITOR system program will be started. The connections concerned are indicated in the drawing of part of the keyboard in figure 8. Row X3 corresponds to pin 36 of the keyboard encoder IC, IC1 (AY-5-2376), and column Y1Ø corresponds to pin 21 of the same IC.

- Link up points b and c and connect one contact of the BREAK key to ground – see figure 9.
- The keyboard does not indicate these connections (next to IC1) by name, as there was insufficient room on the component overlay. This

has, however, been done in figure 9. In addition, the corresponding connections to the Elekterminal are indicated. Make sure that S8, and not S6, of the keyboard is connected to KB5 of the Elekterminal. This automatically generates the ASCII code of the corresponding capital letter whenever a letter key is operated, without having to use a 'shift' (SFT) key (there are two of them). In any case, the character generator in the Elekterminal can not generate any lower case letters (unless the modification described in the January 1981 issue of Elektor is carried out). Do not forget that the Elekterminal requires a –12 V supply as well as a +5 V supply.

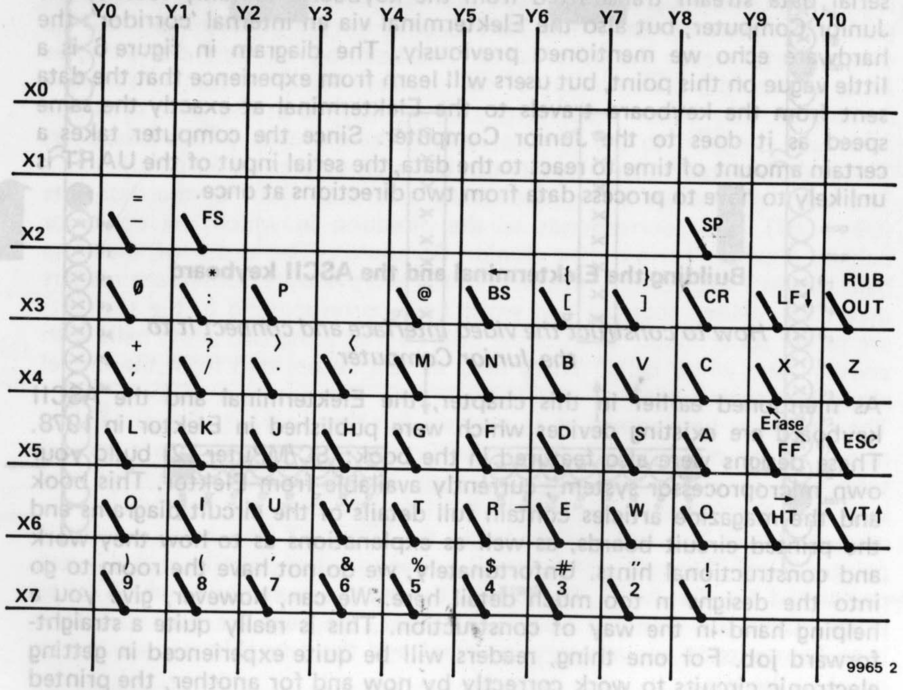


Figure 7. The key matrix of the ASCII keyboard. The RES key now performs the function RUBOUT.

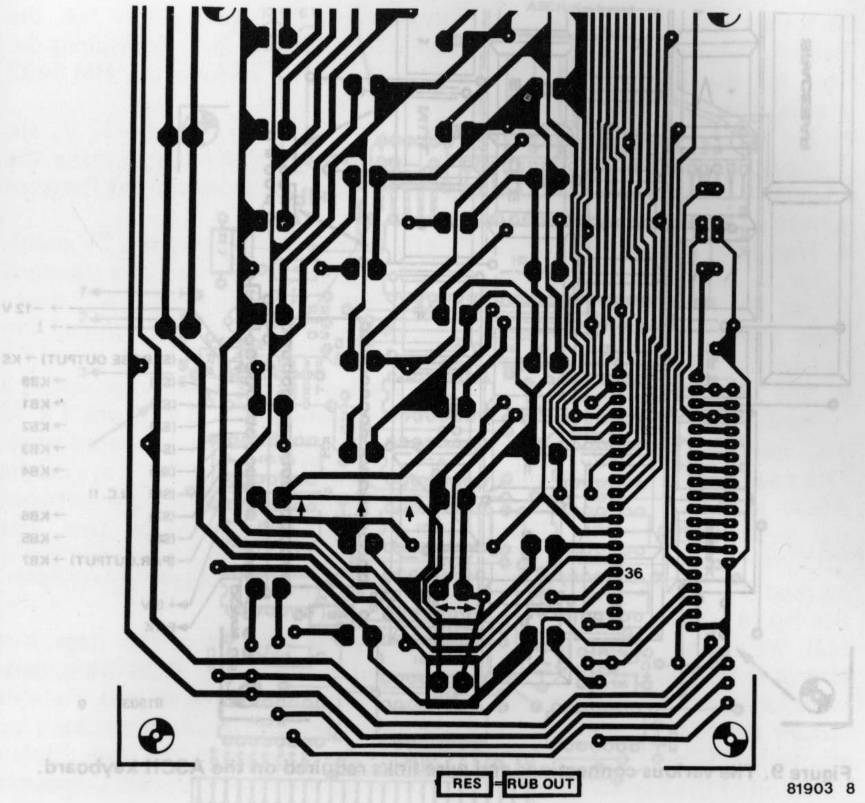


Figure 8. The connections required for the RES key to perform its new (RUBOUT) function.

The Elekterminal

The section of circuit diagram in figure 5 contains all the information necessary to modify the Elekterminal for use with the Junior Computer. The (reduced) component overlay for the Elekterminal can be seen in figure 10. This includes a few suggestions as to where to place the various wire links. Now for the details:

- First of all, diode D4 has been removed, as it is no longer necessary.
- The Baud rate switch, S2a/S2b, is omitted. The Elekterminal now operates at a fixed speed of 1200 Baud (= 1200 bits per second). The

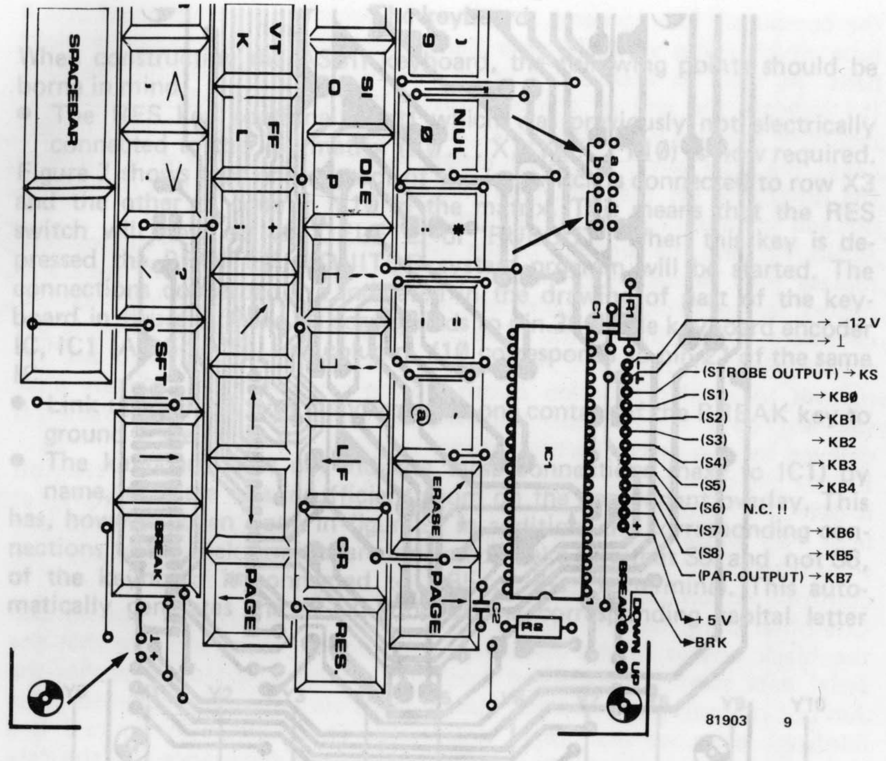


Figure 9. The various connections and wire links required on the ASCII keyboard.

wiper contact, MS2b, is connected directly to ground and points 'MS2a' and '1200' are linked to each other – see figure 10.

- The full/half duplex switch, S1, may be replaced by a wire link between points 'U' and 'V'. If S1 is maintained on the board, the switch will have to be permanently closed. This allows an 'echo' of the depressed key to appear on the video screen.
- The wire links around points 35 . . . 39 are made according to the choice between parity/no parity, odd/even parity, one/two stop bits and 5/6/7/8 bits per character. The connections for the preferred ASCII code (transmitted parity bit, even parity, two stop bits and seven bits per character) are shown. Although the PM program does not carry out a parity check when characters are received, the parity bit will have to be transmitted for other reasons. In figure 10 the centre connections for links 36 and 37 have been omitted for the sake of clarity. These points are connected to +5 V.
- If no extra pages are added to the memory of the Elekterminal, there will be thirteen connections between the Elekterminal and the ASCII keyboard. These are indicated in figure 10. The connections can be made via suitable plugs and sockets or can be 'hard-wired' with ribbon cable.

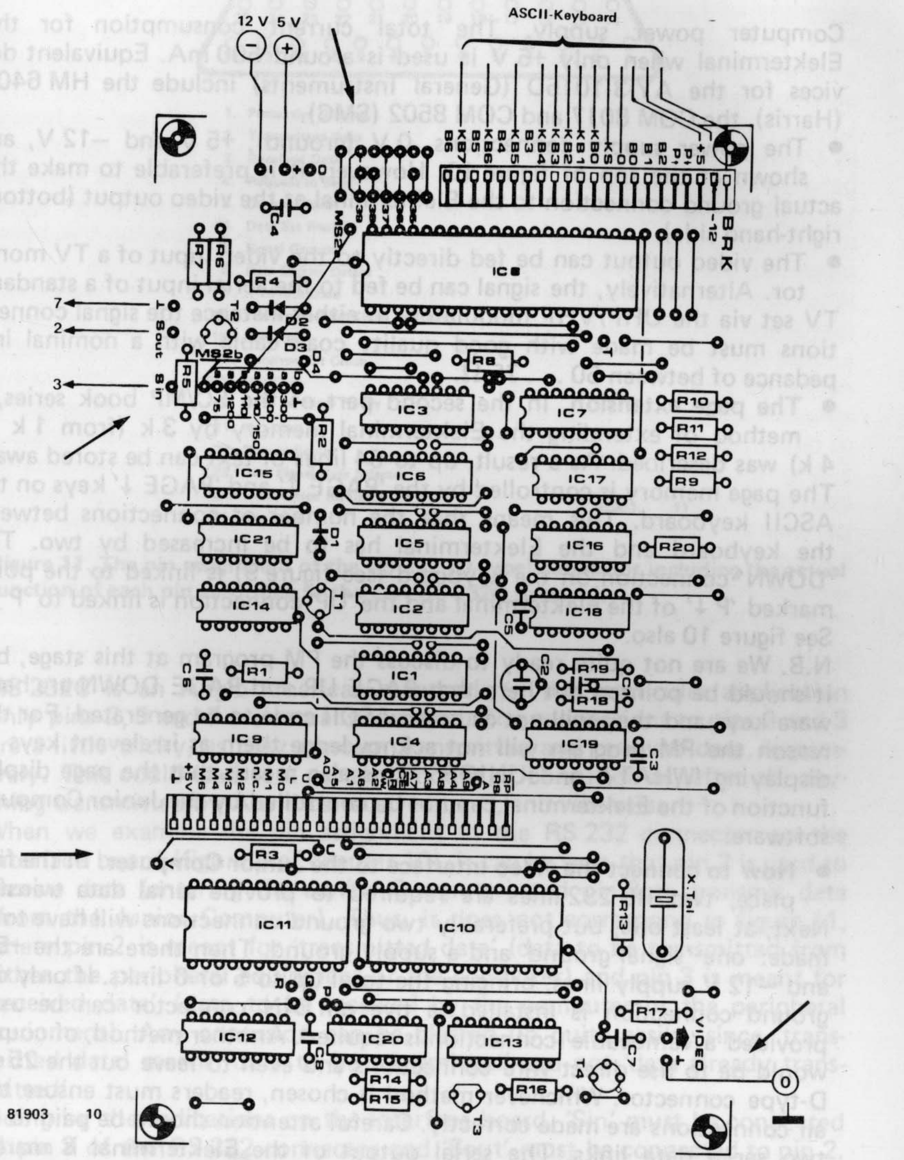


Figure 10. The various connections and wire links required on the Elekterminal (video) board.

Choice of UART (IC8)

Two versions (at least) of the UART used by the Elekterminal exist. The AY-5-1013A, which requires +5 V and –12 V supplies and the AY-3-1015D, which only requires a +5 V supply. The latter is obviously preferable as the current requirements for the –12 V supply will then be reduced. The Elekterminal can then be fed from the modified Junior

Computer power supply. The total current consumption for the Elekterminal when only +5 V is used is around 500 mA. Equivalent devices for the AY-3-1015D (General Instruments) include the HM 6402 (Harris), the COM 8017 and COM 8502 (SMC).

- The power supply connections, 0 V (ground), +5 V and −12 V, are shown at the top of figure 10. However, it is preferable to make the actual ground connection to the Elekterminal at the video output (bottom right-hand side).
- The video output can be fed directly to the video input of a TV monitor. Alternatively, the signal can be fed to the aerial input of a standard TV set via the UHF/VHF modulator. In either instance the signal connections must be made with good quality coax cable with a nominal impedance of between 50 . . . 75 Ω.
- **The page extension.** In the second part of the SC/MP book series, a method of extending the Elekterminal memory by 3 k (from 1 k to 4 k) was described. As a result, up to 64 lines of text can be stored away. The page memory is controlled by the 'PAGE ↑' and 'PAGE ↓' keys on the ASCII keyboard. This means that the number of connections between the keyboard and the Elekterminal has to be increased by two. The 'DOWN' connection on the keyboard (see figure 9) is linked to the point marked 'P ↓' of the Elekterminal and the 'UP' connection is linked to 'P ↑'. See figure 10 also.

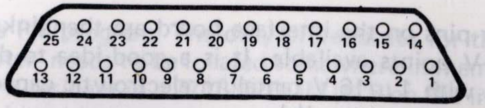
N.B. We are not quite ready to discuss the PM program at this stage, but it should be pointed out here that PAGE UP and PAGE DOWN are hardware keys and they will not cause an ASCII code to be generated. For this reason the PM program will not acknowledge them as irrelevant keys by displaying 'WHAT?' or 'JUNIOR'. This also means that the page display function of the Elekterminal can not be controlled by the Junior Computer software.

- **How to connect the video interface to the Junior Computer.** In the first place, two RS 232 lines are required to provide serial data transfer.

Next, at least one, but preferably two ground connections will have to be made: one 'signal ground' and a supply ground. Then there are the +5 V and −12 V supply lines, bringing the total up to 5 or 6 links. If only one ground connection is installed, a five pin DIN connector can be used, provided a 'removable' connection is required. Another method, of course, would be to use direct wire connections and even to leave out the 25 pin D-type connector. Whichever method is chosen, readers must ensure that all connections are made correctly. Careful attention should be paid to the two serial data links. The serial output of the Elekterminal is marked 'Sout' on the printed circuit board and the serial input is marked 'Sin'.

Let us go back to figure 2 in chapter 10. The point marked 'Sout' on the Elekterminal printed circuit board is connected to pin 2 of the RS 232 connector on the interface board. Point 'Sin' on the other hand is connected to pin 3 of the RS 232 connector. In both cases, the links can be made either directly to the board(s) or by using suitable cable terminated with male D-type connectors at each end. The latter is by far the best method of connecting the two devices to each other.

Figure 11 shows the pin assignment for a 25 pin male D-type connector. Below this a clear survey of the various connections is given. The EIA



1. Protective Ground
2. Transmitted Data
3. Received Data
4. Request to Send
5. Clear to Send
6. Data Set Ready
7. Signal Ground
8. Data Carrier Detect
9 . . . 14 Not Used
15. Transmitted Bit Clock Internal
16. Not Used
17. Received Bit Clock
18,19. Not Used
20. Data Terminal Ready
21. Not Used
22. Ring Indicator
23. Data Signal Rate Selector
24. Transmitted Bit Clock
25. Not Used

81903 11

Figure 11. The pin assignment of the RS 232 (D-type) connector including the actual function of each pin according to the EIA RS 232C standard.

RS 232C is an Anglo-American standard. In this particular application only pins 2, 3 and 7 are actually involved. The connections to pins 2 and 3 are a little tricky, as they concern transmitted and received data, respectively. This could lead to a certain amount of confusion as, after all, anything transmitted by A will be received by B, and vice versa.

When we examine the pin assignment of the RS 232 connector on the interface board (figure 2 of chapter 10), it can be seen that pin 2 is used to receive data (entering the Junior Computer) and pin 3 to transmit data (from the Junior Computer). Thus, it does not correspond to figure 11, where pin 2 is meant for 'transmitted data' (data to be transmitted from either the peripheral equipment or the computer) and pin 3 is meant for 'received data' (data to be received by the computer or the peripheral equipment). Any confusion can be cleared up quite easily, since 'transmitted data' means 'data to be transmitted' and not 'data already transmitted'.

Going by the indications on the interface board, 'Sin' must be connected to pin 3 of the RS 232 connector and 'Sout' must be connected to pin 2. The ground (common) connection should be made to pin 7 of the D-type connector.

Power supply connections. It has already been mentioned that the Elekterminal and ASCII keyboard can be powered from the 'revised' Junior Computer power supply. However, it is also feasible to use a separate power supply. This alternative makes the Elekterminal universally compatible and then, of course, no supply links are necessary.

If required, two of the unused pins (9 . . . 14) of the D-type connector (see figure 11) can be used for the +5 V and −12 V connections. The corre-

sponding solder pins on the interface board are then linked to the nearest +5 V and −12 V points available. It is a good idea to decouple the two supply voltages with $1\,\mu$/16 V tantalum electrolytic capacitors (watch the polarity!) at the connector end(s).
During the description of the constructional details of the interface board in chapter 10, the fact was mentioned that wire links should be placed between pins 4, 5 and 8 and between pins 6 and 20 of the RS 232 connector. These links are not required when the Elekterminal is used. They were mentioned in chapter 10 because, once the D-type connector has been mounted, they are very difficult to install.
Why and when are these links required? They are needed if another peripheral device is used instead of the Elekterminal (a large printer with a built-in keyboard, for instance). Such devices communicate with the computer much more extensively. What is being referred to here is the so-called **handshake** system, where each activity must follow a set pattern of questions and answers in both the computer and the peripheral device. The point being that the software and hardware of the Junior Computer do not conform to this system. This means that when such an item of peripheral equipment is employed, some form of 'artificial answer' will have to be incorporated, so that the Junior Computer will continue to operate. In computer terms, what is required is an 'automatic loop'. This is in fact what the above mentioned links are for.

Now that we have our peripheral equipment built and ready to go, the question arises: what can we do with it? Unfortunately, this can not be answered in a single sentence and will, in fact, take up the rest of this chapter.

The PRINTER MONITOR (PM) system program

Control of communications

Actually, the term 'printer monitor' is rather misleading, for the program could equally be termed 'video monitor'. The word 'printer' is usually associated with printing on paper rather than on a TV screen. Although the computer can be made to work with a printer, in most instances a video display (a video interface, such as the Elekterminal, and a domestic television set) will be used. The reason it was not called 'video monitor' was to avoid any confusion between it and the hidden monitor systems in such places as banks and department stores, etc.!
The PM program constitutes more elaborate alternative to the original monitor program. It enables a great deal of information to be observed at the same time. Not only what is being processed at the moment, but also what went on before, so that the programmer has a comprehensive survey of the whole situation. The last sixteen lines are always shown on the TV screen. Furthermore, there are more key functions, including two that are required by the TAPE MONITOR program (the cassette interface software). All these key functions are used to process data in machine language. Earlier on it was mentioned that it is also possible to work in higher level languages such as BASIC and still use the same hardware pertaining to the PM program. The only prerequisite for this is that some form of 'interpreter' needs to be located somewhere in memory.
The PM program is resident in a 2716 EPROM (IC5 on the interface board). This means that it can be run as soon as the Junior Computer is switched on and does not have to be entered from the keyboard or from cassette beforehand. The program occupies the address area 1ØØØ . . . 14F3 (see the hex dump in Appendix 5). As with the TM program, it requires more than 1K of memory space, but less than 2K, so there is sufficient room for any additional software that the user may wish to add at a later date. This can be accomplished by simply reprogramming the device. The EPROM containing the PM program has to be placed in the socket for IC5. It is no good placing it in the socket for IC4, modifying the start address and then hoping for the best! From chapter 10 we know that IC5 can also be a 1K RAM device. Obviously, this will not meet our requirements and so a 'resident' solution is much more preferable. Some readers may like to program the 2716 themselves using the hex dump given in Appendix 5. The contents of the EPROM can then be checked with the aid of the original monitor program (enter AD, 1ØØØ, +, + etc. and examine each byte).
The original monitor program contains a number of subroutines which can be utilised by the user program. Examples are routines which are used to display certain information, or to wait for a key to be depressed and then identify it. The same is true of the PM program. It includes a subroutine to transfer an ASCII character to the video interface or to the printer. Another reads in characters transmitted from the keyboard (thus, it waits for a key to be depressed). Then there are all sorts of control function routines: enter a space, start a new line, etc. The various routines are fully described in Book 4, in the chapter devoted to the PM software.
Under supervision of the PM, a user program may be run in one go or in the step mode. We know this feature from the original monitor. Now for the interesting bit: A single key operation is enough for the computer to display the contents of all the internal registers at the same time. This helps to keep the operator fully informed about the various intermediate phases during program execution.
The time has come to get to know the PM program a little better, which can be accomplished with the aid of figure 12. This shows the ASCII keyboard and the keys used by the PM program.

① RES = RUBOUT

The RES(ET) key on the ASCII keyboard is used to RUBOUT information. Depressing this key will cause the ASCII character 7F (seven ones in a row) to be transferred from the keyboard to the Junior Computer by way of the UART. This is very important when the PM program is to be started for the **second** time. Well, for there to be a second time, there has to be a first! The first start takes place via the original monitor routine:
(switch on)
RST 1 Ø Ø Ø GO
The monitor of the PM program has been started up. The ignition key has

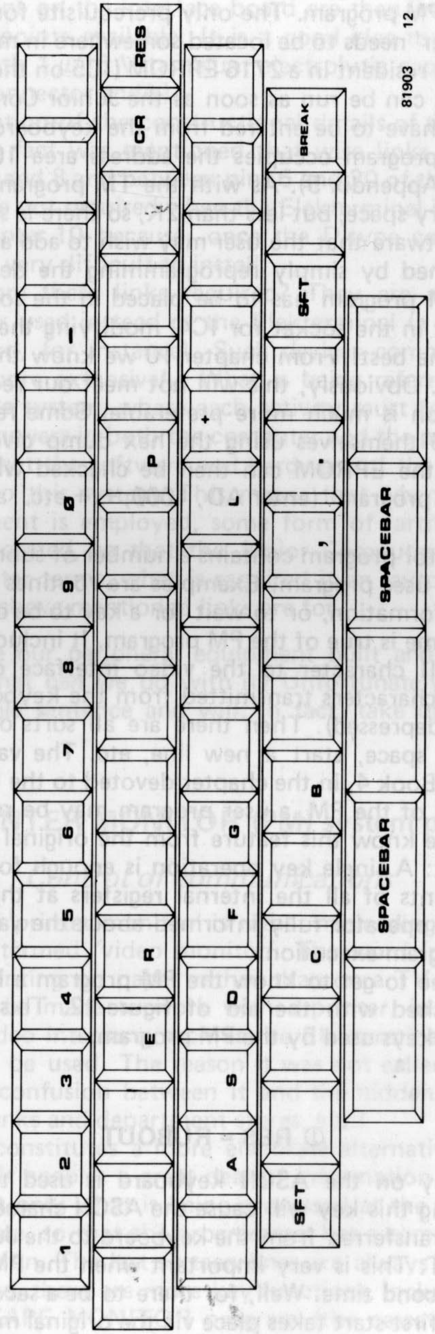


Figure 12. The keys on the ASCII keyboard which are used by the PRINTER MONITOR program.

been turned, so to speak. The start section of the PM program is executed. Now the computer waits until the PM car is actually ready to drive. A kind of 'warm' start takes place:

RUBOUT (= RES)

As a result, the **start character, 7F,** is sent to the computer. This enables the rest of the PM program to be dealt with. The computer determines the speed (Baud rate) at which the start character was transmitted. All the characters transmitted by the computer will be sent at the same speed as this. When the required time measurements have been taken, the computer reports back:

JUNIOR

blank line

This reaction is called a 'prompt'. Thus, the text 'JUNIOR' is printed and then two commands to start a new line are given (Carriage Return = CR plus Line Feed = LF). This gives an empty space between the word JUNIOR and the text to follow.

After this the PM program is ready for use. Now it is simply a question of waiting for an instruction to be entered, so a key or several keys will have to be depressed.

N.B. If the Elekterminal is not used, the start character 7F can usually be obtained by depressing 'control' and 'DEL' (= delete) simultaneously.

② The alphanumeric keys Ø . . . 9 and A . . . F

③ SP (space)

④ '.' (fullstop)

This concerns the most elementary keys and key functions. The alphanumeric keys Ø . . . 9 and A . . .F are used in exactly the same way as in the original monitor program. In other words, they constitute data, address information and program numbers etc. The full hexadecimal repertoire. Special input buffers, locations INH and INL (see table 1), are used to enter data or address information. When a **work address** and associated data is entered, however, the operation is quite different to that when the original monitor is used (AD, DA, + and alphanumeric keys):

- First the alphanumeric data is entered, then the user tells the computer whether the data is a work address or data belonging to that memory location.
- If a work address is involved, SP (space bar) is operated.
- If data is to be entered at a work address (which will have been typed in previously), depress the '.' key.
- If an address or data starts with Ø, this may be omitted. The work address ØØ2ØØ can be entered as 2ØØ. If SP is depressed, the work address will be ØØØØ. The work address is loaded with ØØ (the opcode of the BRK instruction, for instance) by depressing the '.' key. As soon as any alphanumeric data has been processed the input buffers INH and INL are reset (ØØ).
- As soon as data has been entered by way of the '.' key, the current work address is incremented by one and is printed on a new line along with its corresponding contents. In other words, there is no need to depress the plus key as with the original monitor. This saves a lot of time when entering a program, which after all is simply a sequence of addresses. Let us take as an example the addition program on page 62 of Book 1.

Table 1. The various RAM locations used by the PM program.

TEMPORARY DATA BUFFERS			
TEMP	*	$ØØFC	
TEMPX	*	$ØØFD	
STPBIT	*	$1A59	NUMBER OF STOP BITS
CNTL	*	$1A5A	BIT TIME BUFFER
CNTH	*	$1A5B	
CNTHL	*	$1A5C	HALF BIT TIME BUFFER
CNTH	*	$1A5D	
TIML	*	$1A5E	COUNT DOWN BUFFER
TIMH	*	$1A5F	
TEMPA	*	$1A6Ø	TEMPS
TEMPB	*	$1A61	
CHA	*	$1A62	CHARACTER BUFFER
PARAL	*	$1A63	PARAMETER BUFFERS
PARAH	*	$1A64	
PARBL	*	$1A65	
PARBH	*	$1A66	
PRTEMP	*	$1A67	TTY BUFFER
BRKT	*	$1A7C	BREAK TEST VECTOR
* * * BUFFERS & EXTERNAL ADDRESSES * * *			
DISCNT	*	$1A68	DISPLAY COUNTER
COLDST	*	$1CB5	EDITOR COLD START
WARMST	*	$1CCA	EDITOR WARM START
BEGIN	*	$1ED3	EDITOR SUBROUTINE
RESET	*	$1C1D	RESET OF VERSION D
GETKEY	*	$1DF9	COMPUTE THE KEY VALUE
LDAINH	*	$1DA7	PART OF SCAND/SCANDS
BEGADL	*	$ØØE2	BEGIN ADDRESS POINTER
BEGADH	*	$ØØE3	
ENDADL	*	$ØØE4	END ADDRESS POINTER
ENDADH	*	$ØØE5	
CENDL	*	$ØØE8	CURRENT END ADDRESS POINTER
CENDH	*	$ØØE9	

First let us define the IRQ vector (which is pointing to the start address of a SAVE-like routine in the PM program, but we shall come back to that later) in connection with the BRK instruction. Then the program data is entered.

Imagine that the PM program is already running and that the JUNIOR prompt has appeared on the video display.

```
( 1)  1A7E  (SP)
( 2)  1A7E  XX
```

After depressing the alphanumeric keys 1, A, 7 and E and then operating the space bar, (SP), the Junior Computer will react by printing the work address, a space, the data contained in the work address and then another space.

Now for a few practical considerations. The keystrokes of the operator are displayed with a normal type of letter, including '(SP)' and other 'invisible' commands (they are not entirely invisible since the position of the cursor will indicate when a CR, LF, SP etc. has been operated). The response by the Junior Computer is printed in **bold type**. The figure in parentheses at the beginning of a line is the line number. It has been included here for editorial reasons, but is not actually displayed in practice.

Right, so a work address has been entered and we wish to store the data 'CF' in it. Thus:

```
( 2)  1A7E  XX   CF.
( 3)  1A7F  XX
```

The next address therefore appears on the screen and is to contain the data '14':

```
( 3)  1A7F  XX    14.
( 4)  1A8Ø  XX
```

Address 1A8Ø is irrelevant in this particular instance and so the program can be keyed in from address Ø1ØØ. We will therefore have to change to a new work address:

```
( 4)  1A8Ø  XX   1ØØ  (SP)
( 5)  Ø1ØØ  XX
```

and then we can type in the required data:

```
( 5)  Ø1ØØ  XX   18.
( 6)  Ø1Ø1  XX   A9.
( 7)  Ø1Ø2  XX   13.
( 8)  Ø1Ø3  XX   69.
( 9)  Ø1Ø4  XX   Ø8.
(10)  Ø1Ø5  XX  .
(11)  Ø1Ø6  XX
```

A full stop was entered after address Ø1Ø5 so that the BRK instruction (opcode = ØØ) is copied into that location. The entire program has now been transferred to the memory banks of the Junior Computer.

⑤ The '+' key
⑥ The '–' key

These functions are virtually self-explanatory and can be described very briefly. The '+' key produces an address which is one higher than the one previously on display (in other words, it increments the current address by one). The – key, on the other hand, produces an address which is one lower than that previously on display (it decrements the current address by one). In both instances, the new work address is printed together with the data contained therein. These keys are of invaluable assistance when sections of the memory need to be checked, such as the newly entered addition program for instance:

```
(11)  Ø1Ø6  XX  +
(12)  Ø1Ø7  XX
```

Oops, wrong way!

```
(12)  Ø1Ø7  XX  –
(13)  Ø1Ø6  XX  –
(14)  Ø1Ø5  ØØ  –
(15)  Ø1Ø4  Ø8  –
(16)  Ø1Ø3  69  –
```

```
(17)  Ø1Ø2   13  —
(18)  Ø1Ø1   A9  —
(19)  Ø1ØØ   18
```

which brings us back to the start address. All the relevant data has now been entered correctly and the program can be started.
N.B. The '+' key on the Elekterminal can only be operated in conjunction with the 'shift' key (SFT).

⑦ The 'R' key

The 'R' stands for 'RUN' and is identical to the GO key of the original monitor program. The operation of this key allows a program to be carried out starting from the work address currently on display. The program may either be run in one go or can be run in the step mode, depending on whether the STEP switch has been operated.
Let us continue with the addition program:

```
(19)  Ø1ØØ   18   R
(20)  Ø1Ø7   XX
```

What does line 20 means? The IRQ jump vector is pointing to address 14CF. As far as the PM program is concerned this location is similar to 1CØØ for the original monitor program. In the event of a non-maskable interrupt (NMI) or an interrupt request (IRQ), a short interrupt routine at the start of the PM program is dealt with. This ensures that various program data is saved and also sees to it that the current program counter and its contents (the following start address) is displayed on the screen. The BRK instruction causes an address to be 'skipped' (see page 101 in Book 2). This is why line 20 contains address Ø1Ø7 instead of the expected address of Ø1Ø6.

⑧ The 'P' key

This key performs the same function as the PC key in the original monitor program. Depressing the P key when a program is being run in the step mode, allows the programmer to continue where he/she left off after temporarily interrupting the program to check on various parameters (by depressing L = List for instance – see ⑨). The return address after such an interruption is printed on a new line after the P key is operated and acts as a new work address. Then the R key should be operated to carry out the next step (instruction).

⑨ The 'L' key

As mentioned above, L stands for 'List' and when this key is depressed the contents of all the internal registers of the 6502, which are stored in RAM during the interrupt routine at the start of the PM program, are displayed on the video screen.
The complete function can best be explained with the aid of an example.
Let us continue from line 20 of the addition program:

```
(20)  Ø1Ø7   XX   L
(21)  ACC:   1B
(22)     Y:  XX
(23)     X:  XX
(24)    PC:  Ø1Ø7
(25)    SP:  Ø1FF
(26)    PR:  ØØ11Ø1ØØ
(27)         NV BDIZC
```

The C in line 27 is followed by a space. As can be seen, the whole procedure is much swifter and more efficient than that of the original monitor. Everything can be seen at a single glance rather than by one location at a time (from ØØEF to ØØF5). Neither is there any need to spell out the names of the flags in the status register (PR) as they all appear beneath the actual contents of the latter. The result of the addition (1B) can be seen in the accumulator (ACC) immediately. The X and Y registers are omitted from the picture altogether and contain XX here, as usual when the contents are of no importance. The contents of the program counter (PC) are Ø1Ø7, which is quite correct due to the BRK instruction. The contents of location SPUSER are Ø1FF, which is what it contained at the beginning of the PM program. This is also correct as nothing was stored on or removed from the stack during the addition program and no subroutines were involved. The B flag will be set as we have just witnessed a BRK instruction and the N, Z and C flags will al be reset (Ø) because the result of the addition is positive and smaller than FF.
Thus, L is a very useful key.

⑩ The 'M' key
⑪ The ',' key
⑫ The 'CR' key

Perhaps this is not so obvious, but M stands for 'Matrix'. The term matrix is used here to imply rectangle made up of various data. A normal rectangle features a certain length and a certain width. A matrix, however, features a number of rows and a number of columns. Individual data occupies a specific row and a specific column. A data matrix is in fact none other than the well known **hex dump.** A hex dump is a data matrix consisting of sixteen columns and (in this instance) any number of rows. The actual number of rows depends on the amount of data to be dumped. Divide this amount of data by sixteen, add one to it if a remainder is left and the result will be the number of rows in the hex dump. The last row need not be completely filled with data.
The M key (together with keys S and G) belongs to a set of functions that have **parameters** attached to them. This means that when these keys are operated certain extra details will have to be entered as well. If more than one parameter is involved, the relevant data will have to be separated by a comma. The key function will then be carried out when the operator presses the CR (Carriage Return) key – to take the cursor back to the beginning of the next line.
Which parameters are involved when a hex dump needs to be printed? Obviously, the first and last address of the data block to be dumped. How this is accomplished can best be illustrated by using the previous program example.
Before the above 'interrupt' we had reached line 27:

```
(27)         NV BDIZC   M
(28)  HEXDUMP:
```

The first address of the data block is then entered, the ',' key depressed, the last address of the block entered and finally CR is depressed. Again, the leading zeros can be ignored. The first address entry is completed by the operation of the '.' key and **must be lower than the last address.** If this is not the case, an error will be reported. We will come back to this later, but first:

```
(28)  HEXDUMP:   1ØØ,   1Ø5   CR
(29)             Ø   1   2   3   4   5  6  7  8  9  A   B   C   D   E   F
(30)  Ø1ØØ: 18  A9  13  69  Ø8  ØØ
(31)  (blank line)
(32)  JUNIOR
(33)  (blank line)
```

a new line is started: line 34.

First the column numbers (Ø . . . F) are printed. The next line starts with the address of the first two bytes of data to be dumped. This data then follows under the corresponding columns. The hex dump of the program example does not occupy a complete row, but obviously, if the data block were any larger the dump would have taken up several rows.

One thing we have to watch out for is the fact that the address of the first line of the hex dump does not necessarily have to be XXXØ (where X = Ø . . . F). It all depends on what the user has selected. However, he/she is strongly recommended to start the dump at XXXØ, as then the correct data will fall into the correct rows and columns. This was the case in the example (lines 29 and 30) that we have just given.

⑬ The 'S' key

In actual fact, this key corresponds to the SAVE function of the TM program (chapter 11). Thus, the PM program can also be used to store data blocks or programs on cassette. Again, the DUMP/DUMPT subroutine is involved and as this belongs to the TM program, the latter will have to be available before the S key can be used.

After the S key has been operated, the file number, ID, the start address, SA, and the end address, EA (= LA + 1) have to be entered. The moment at which the actual SAVE function is executed, by depressing the CR key, corresponds to the moment at which the GET key is depressed during the TM program. Part of the necessary preparations were discussed in chapter 11.

Program numbers ØØ and FF should not be generally used. To do so would cause an error report.

Here is a practical example. The addition program:

```
(34)  S11,   1ØØ,    1Ø6 CR
(35)  READY
```

Line 35 is printed as soon as the data block has been transferred to tape. The next activity on the part of the user will be printed on line 36. The addition program has been given the file number 11. All the data between SA and EA (up to and including LA) is written onto the tape.

⑭ The 'G' key

Just as GET is twinned with SAVE in the TM program, so the G key is related to the S key in the PM program. The G key must be depressed to read a data block from tape during the PM program. Clearly, the RDTAPE subroutine of the TM program will have to be available, otherwise key G will have no function to perform.

One parameter which must always be indicated is the file (or program) number, ID. If the program number FF is chosen, the start address will have to be entered (see chapter 11). If FF is indicated the first data block to arrive will be read and stored in memory starting from location SA.

Another practical example. The addition program was found to be so useful that is was stored on cassette (lines 34 and 35), so that it could be used at a later date without having to key in the required six bytes. Now it is needed again and so it is retrieved from tape:

```
(36)  G11 CR
(37)  READY
```

Line 37 is printed as soon as the data block with a file number of 11 has been found and completely loaded into memory in the address range Ø1ØØ . . . Ø1Ø6. Now it can happen that the routine needs to be included in a different, more extensive program starting at address Ø2ØØ. This means that the addition program will have to be moved elsewhere. There is plenty of room for it on page zero (from address ØØ1Ø on) and it does not contain any address operands which require modification.

Now what? The tape is rewound until just before the start of file 11, then the cassette recorder is set to the play mode during line 39!

```
(38)  G    FF
(39)  SA:    1Ø   CR
(40)  READY
```

The addition program has been copied into address locations ØØ1Ø . . . ØØ16. This can be checked by calling up the hex dump routine by way of the M key.

⑮ The BREAK key

Supposing, for instance, a hex dump is being printed in the range ØØØØ . . . FFFF and you suddenly realise that the range should have been ØØØØ . . . ØØØF. The first dump will take 4096 times as long as the second, which is rather a long time to wait! It would be nice, therefore, if the procedure could be interrupted in an elegant manner, allowing the computer to a certain point (= start address of a subroutine) in the program. It is rather drastic to resort to switching off the power. Depressing the RST key would lead us back to the original monitor program, which is not what we want either. What about the ST key of the original monitor? Well, this results in a non-maskable interrupt. An indirect jump takes the computer to locations 1A7A and 1A7B, where the NMI jump vector has been defined by the user. Usually, the program will continue from the original monitor (1CØØ).

In the case of the BREAK key, something similar will happen, albeit without involving a non-maskable interrupt. Looking at the section of the Elekterminal circuit diagram in figure 5 we can see that one side of the BREAK key is connected to ground (or at least . . . it should be!) and the other side of the switch is connected to the buffered serial output of the

UART. Depressing the BREAK key causes the normally high serial output line (and port line PA7 of the Junior Computer) to change to logic zero.

As we mentioned previously, the PM program includes a subroutine which has the task of sending an ASCII character to the peripheral device. This subroutine is called PRCHAR. During the final phase of this routine the computer checks to see whether the port line PA7 has gone low. If so, it waits until PA7 has gone high again, in other words, until the BREAK key has been released. After this, an indirect jump to locations 1A7C and 1A7D takes place:

1A7C = ADL, and
1A7D = ADH

These locations contain the **BRK jump vector.** Right at the beginning of the PM program, the BRK vector is loaded with the start address of LABJUN, location 1Ø5F. The section of program starting from this address makes sure that the text JUNIOR is displayed on the screen and that the stack pointer (SP) and the contents of location SPUSER are made equal to FF. It is possible to define a different BRK vector, but that will have to be accomplished during the PM program. Note: The BRK jump vector referred to here has nothing to do with the IRQ vector during a BRK **instruction** – it is something quite different!

N.B. From the above it can be seen that the BREAK key is only active during the transmission of an ASCII character by the Junior Computer. Thus, while the PM program is still waiting for a new key to be depressed, operating the BREAK key will have no effect whatsoever.

Right, that just about covers all the relevant keys on the ASCII keyboard which are utilised by the PM program. Depressing any other key will merely cause a surprised 'WHAT?' to appear on the screen. Either that, or the PM program answers 'JUNIOR' which does not help matters either! Now let us play around with the R, P and L keys.

Using the PM program to step through a program

As mentioned earlier. The PM program can also be used to carry out user programs step by step, one instruction at a time. The procedure is as follows:

- Make sure that the hardware on the main board is adapted correctly. By this we mean that the module discussed in chapter 10 should be mounted: see figures 8b, 9, 11 and 12 in that particular chapter. The two select lines (K) must be linked to the corresponding pins of IC6 to prevent programs in the address range 1ØØØ . . . 17FF (select lines K4 and K5 are linked to enable the EPROM containing the PM program) and in the address range 1CØØ . . . 1FFF (K7) from being run in the step mode. In one instance the address range selected by K6 must disable a non-maskable interrupt to allow for a small monitor extension in PIA RAM on page 1A. This concerns both the original monitor (K6 and K7, see Appendix 2) and the PM program (K4, K6 and K7) during a decimal arithmetic operation (see later on in this chapter).
- Make sure that the STEP switch (S24) is ON. The red LED in the GO key will then light.
- The NMI vector need not be entered (except during a decimal arithmetic operation). This is automatically pointing to address location 14CF (STEP label) at the beginning of the PM program. This address marks the start of a kind of SAVE routine. All the 6502 registers are examined and stored in the familiar RAM locations on page zero. In addition, the address and its contents belonging to the opcode of the next instruction to be executed are printed on the screen.

A practical example

Chapter 3 in Book 1 gave a few examples of how to step through a user program. Although it is the PM program involved here and not the original monitor program, the two routines have a great deal in common and so a lot of what was said in chapter 3 can be applied here. It may be a good idea to read that particular chapter again and so refresh your memory.

Now for an example. Look at the following series of figures very closely:

```
Ø     1     3     6     10     15 . .
  +1    +2    +3    +4     +5     + .
```

The difference between the second and the first numbers is 1, that between the third and second is 2, that between the fourth and third is 3, etc. Each time the difference between two consecutive numbers increases by one. We will now write a program to calculate the first eleven numbers of the series, including the first which is zero. The numbers must be saved on the stack in the correct order, starting with zero at location Ø1FF. Remember, this program is only meant to illustrate the step mode and does not have any particular significance other than that.

The program is shown in figure 13 and is called NUMBERS. The program is based on this principle: first the initial value of the Y index register is made Ø1 and the first number in the series (ØØ) is saved on the stack. Next, the program loop after label ADD is run through ten times in succession: it is run through once and then stepped through nine times because of the BNE instruction. The value in the Y register is stored in location TEMPY, the previous number to be calculated is sent to the accumulator (PLA plus PHA) and the contents of TEMPY are added to those of the accumulator. The PHA instruction is required so that the status of the stack pointer is restored. The new number in the series has now been calculated and is placed on the stack (PHA). The contents of the Y register are then incremented by one (INY) to calculate the next number in the series. A test (CPY #ØB plus BNE) will inform the computer whether the next number is to be calculated or not.

Figure 14 shows the status of the stack and the stack pointer for the different values in the Y register. The contents of locations Ø1FF . . . Ø1F5 are used during a hexadecimal calculation. This is what happens in the program in figure 13. The contents of locations Ø1F4 . . . Ø1EA indicate what happens during a decimal calculation. We shall come back to that later.

During the step procedure the program in figure 13 was 'disguised' in the unfamiliar apparel of figure 15. The black balls represent address locations containing an opcode. These are 'half-way' stations at which the train (the step-by-step procedure during a program) stops temporarily. As can be seen, it is a slow train which stops everywhere. When a program is run

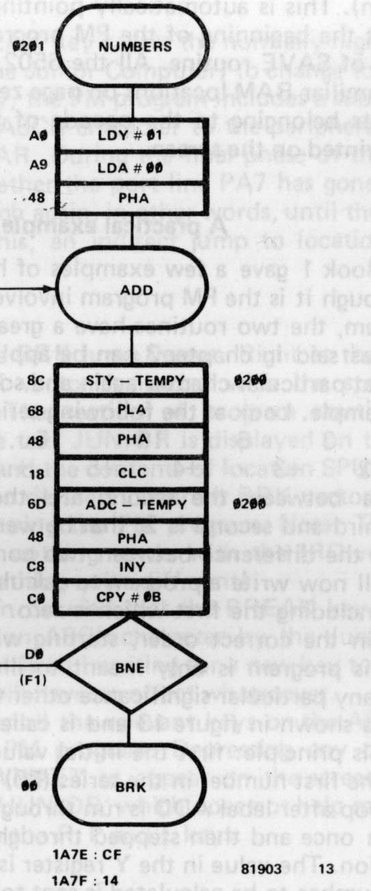


Figure 13. The NUMBERS program performs the task of storing a sequence of values on the stack in the correct order.

through in one go, the train is more like an express. In fact it only stops once during DECAR at address Ø217. The last part of the journey involves the BRK instruction at address location Ø215.

We have been introduced to the NUMBERS program in the form of the detailed flowchart in figure 13 and in the form of the railroad journey in figure 15. A third representation will now be provided, which is not altogether new; the assembler version:

```
NUMBERS  Ø2Ø1  AØ  Ø1      LDY #Ø1
         Ø2Ø3  A9  ØØ      LDA #ØØ
         Ø2Ø5  48          PHA
ADD      Ø2Ø6  8C  ØØ  Ø2  STY-TEMPY
         Ø2Ø9  68          PLA
         Ø2ØA  48          PHA
         Ø2ØB  18          CLC
         Ø2ØC  6D  ØØ  Ø2  ADC-TEMPY
         Ø2ØF  48          PHA
         Ø21Ø  C8          INY
         Ø211  CØ  ØB      CPY #ØB
         Ø213  DØ  F1      BNE ADD
BRK      Ø215  ØØ          BRK
         1A7E  CF
         1A7F  14
```

The first column is clearly intended for labels. Next follows a column of addresses containing an opcode (except for the IRQ jump vector at the end). Then the remaining bytes for each operation are printed; one instruction per line, and finally the mnemonics. If necessary, a further column may be used to express any comments or remarks the operator may wish to make.

It was calculated that stepping through the NUMBERS program involves 93 stops and one final destination at DECAR. If the two termini (NUMBERS and DECAR) are also taken into account, there are a total of

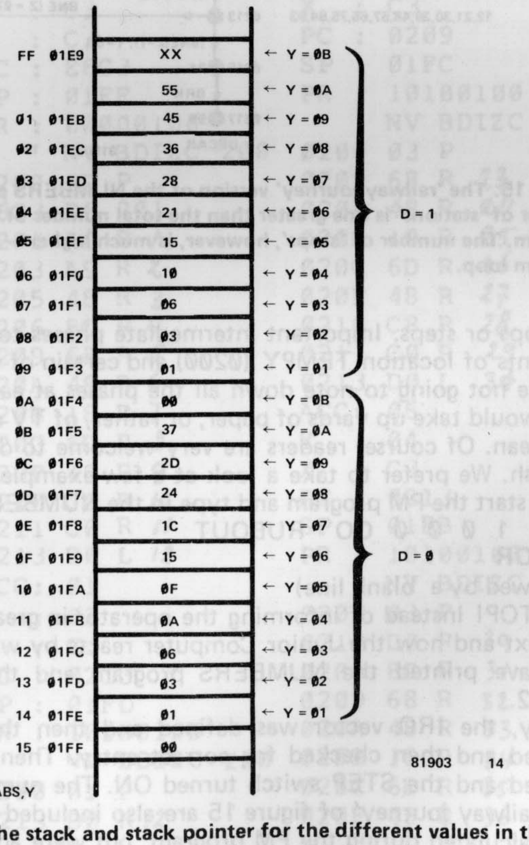


Figure 14. The status of the stack and stack pointer for the different values in the Y index register during the NUMBERS program (figure 13). The left-hand values refer to the PRNUMB program in figure 18.

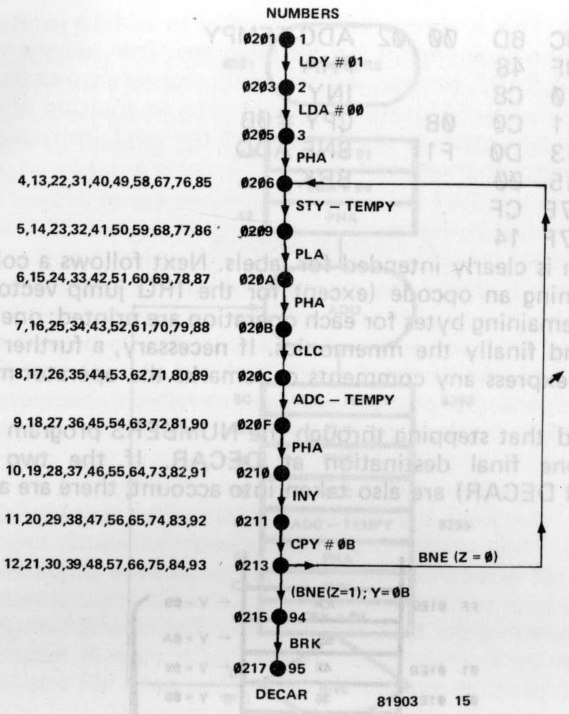

NUMBERS
Ø2Ø1 1
LDY # Ø1
Ø2Ø3 2
LDA # ØØ
Ø2Ø5 3
PHA
4,13,22,31,40,49,58,67,76,85 Ø2Ø6
STY – TEMPY
5,14,23,32,41,50,59,68,77,86 Ø2Ø9
PLA
6,15,24,33,42,51,60,69,78,87 Ø2ØA
PHA
7,16,25,34,43,52,61,70,79,88 Ø2ØB
CLC
8,17,26,35,44,53,62,71,80,89 Ø2ØC
ADC – TEMPY
9,18,27,36,45,54,63,72,81,90 Ø2ØF
PHA
10,19,28,37,46,55,64,73,82,91 Ø21Ø
INY
11,20,29,38,47,56,65,74,83,92 Ø211
CPY # ØB
12,21,30,39,48,57,66,75,84,93 Ø213
BNE (Z = Ø)
Y = Ø2 . . . ØA
(9x)
(BNE(Z=1); Y=ØB
Ø215 94
BRK
Ø217 95
DECAR
81903 15

Figure 15. The 'railway journey' version of the NUMBERS program (figure 13). The number of 'stations' is one greater than the total number of instructions in the program. The number of 'stops', however, is much higher — 94 in all — due to the program loop.

95 stops or steps. Important intermediate phases to keep in mind are the contents of location TEMPY (Ø2ØØ) and certain of the processor registers. We are not going to note down all the phases at each step in figure 15, as that would take up yards of paper, or rather, of TV screen, if you see what we mean. Of course, readers are very welcome to do it themselves if they so wish. We prefer to take a look at a few examples here and there. First, let us start the PM program and type in the NUMBERS program:

RST 1 Ø Ø Ø GO RUBOUT

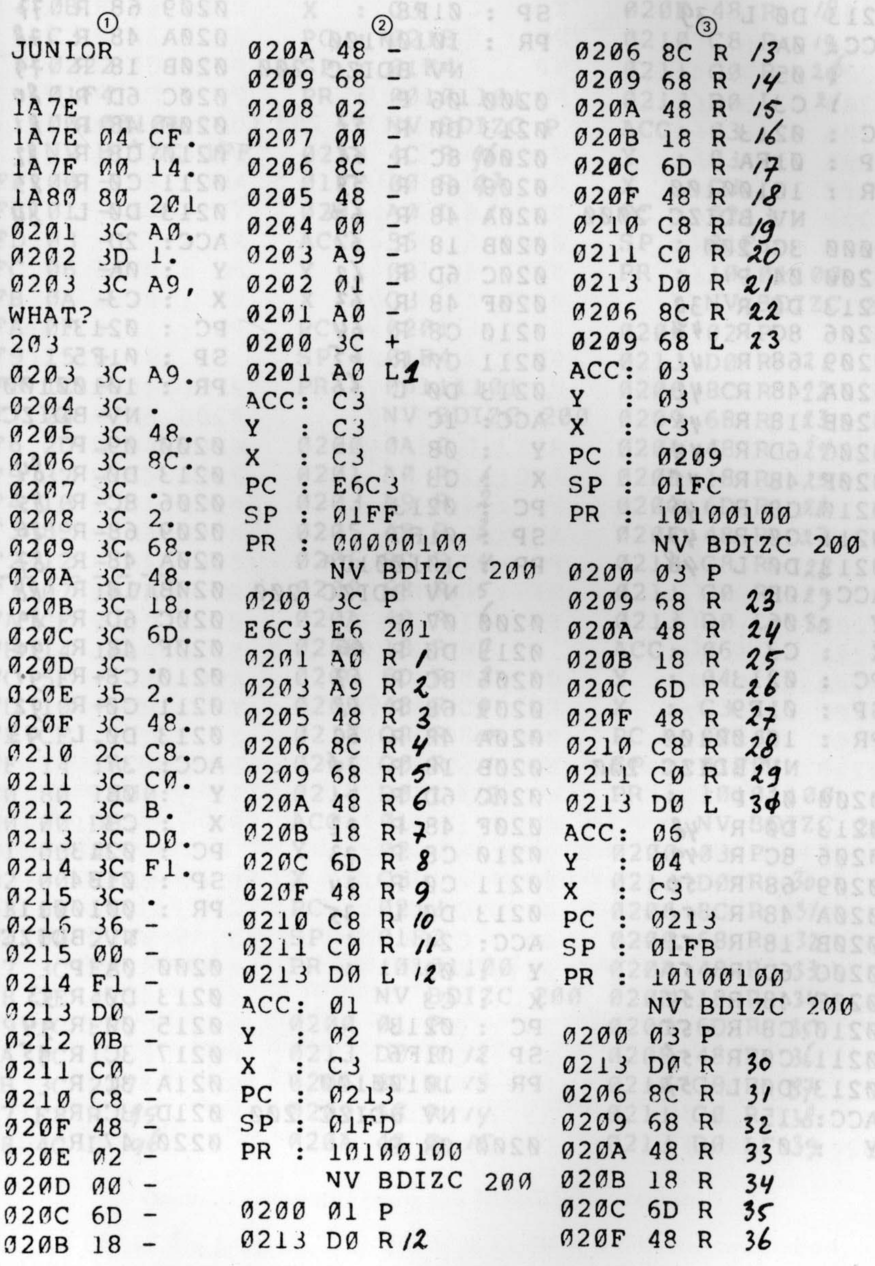
JUNIOR

(followed by a blank line)

. . . STOP! Instead of informing the operator in great detail about what to do next and how the Junior Computer reacts by way of the PM program, we have printed the NUMBERS program and the results obtained in table 2.

Firstly, the IRQ vector was defined and then the actual program was entered and then checked for correct entry. Then the start address was entered and the STEP switch turned ON. The numbers 1 . . . 95 given in the 'railway journey' of figure 15 are also included in table 2. These were not calculated during the PM program, but were added later for the sake of clarity.

Table 2. A printout of the two versions of the NUMBERS program (see figure 13) run in the step mode.



①
```
JUNIOR

1A7E
1A7E 04 CF.
1A7F ØØ 14.
1A8Ø 8Ø 2Ø1
Ø2Ø1  3C AØ.
Ø2Ø2  3D 1.
Ø2Ø3  3C A9,
WHAT?
2Ø3
Ø2Ø3  3C A9.
Ø2Ø4  3C .
Ø2Ø5  3C 48.
Ø2Ø6  3C 8C.
Ø2Ø7  3C .
Ø2Ø8  3C 2.
Ø2Ø9  3C 68.
Ø2ØA  3C 48.
Ø2ØB  3C 18.
Ø2ØC  3C 6D.
Ø2ØD  3C .
Ø2ØE  35 2.
Ø2ØF  3C 48.
Ø21Ø  2C C8.
Ø211  3C CØ.
Ø212  3C B.
Ø213  3C DØ.
Ø214  3C F1.
Ø215  3C .
Ø216  36 -
Ø215  ØØ -
Ø214  F1 -
Ø213  DØ -
Ø212  ØB -
Ø211  CØ -
Ø21Ø  C8 -
Ø2ØF  48 -
Ø2ØE  Ø2 -
Ø2ØD  ØØ -
Ø2ØC  6D -
Ø2ØB  18 -
```

②
```
Ø2ØA 48 -
Ø2Ø9 68 -
Ø2Ø8 Ø2 -
Ø2Ø7 ØØ -
Ø2Ø6 8C -
Ø2Ø5 48 -
Ø2Ø4 ØØ -
Ø2Ø3 A9 -
Ø2Ø2 Ø1 -
Ø2Ø1 AØ -
Ø2ØØ 3C +
Ø2Ø1 AØ L1
ACC: C3
Y  : C3
X  : C3
PC : E6C3
SP : Ø1FF
PR : ØØØØØ1ØØ
     NV BDIZC 2ØØ
Ø2ØØ 3C P
E6C3 E6 2Ø1
Ø2Ø1 AØ R 1
Ø2Ø3 A9 R 2
Ø2Ø5 48 R 3
Ø2Ø6 8C R 4
Ø2Ø9 68 R 5
Ø2ØA 48 R 6
Ø2ØB 18 R 7
Ø2ØC 6D R 8
Ø2ØF 48 R 9
Ø21Ø C8 R 10
Ø211 CØ R 11
Ø213 DØ L 12
ACC: Ø1
Y  : Ø2
X  : C3
PC : Ø213
SP : Ø1FD
PR : 1Ø1ØØ1ØØ
     NV BDIZC 2ØØ
Ø2ØØ Ø1 P
Ø213 DØ R 12
```

③
```
Ø2Ø6 8C R 13
Ø2Ø9 68 R 14
Ø2ØA 48 R 15
Ø2ØB 18 R 16
Ø2ØC 6D R 17
Ø2ØF 48 R 18
Ø21Ø C8 R 19
Ø211 CØ R 20
Ø213 DØ R 21
Ø2Ø6 8C R 22
Ø2Ø9 68 L 23
ACC: Ø3
Y  : Ø3
X  : C3
PC : Ø2Ø9
SP : Ø1FC
PR : 1Ø1ØØ1ØØ
     NV BDIZC 2ØØ
Ø2ØØ Ø3 P
Ø2Ø9 68 R 23
Ø2ØA 48 R 24
Ø2ØB 18 R 25
Ø2ØC 6D R 26
Ø2ØF 48 R 27
Ø21Ø C8 R 28
Ø211 CØ R 29
Ø213 DØ L 30
ACC: Ø6
Y  : Ø4
X  : C3
PC : Ø213
SP : Ø1FB
PR : 1Ø1ØØ1ØØ
     NV BDIZC 2ØØ
Ø2ØØ Ø3 P
Ø213 DØ R 30
Ø2Ø6 8C R 31
Ø2Ø9 68 R 32
Ø2ØA 48 R 33
Ø2ØB 18 R 34
Ø2ØC 6D R 35
Ø2ØF 48 R 36
```

④
```
Ø210 C8 R  37
Ø211 CØ R  38
Ø213 DØ L  39
ACC: ØA
Y  : Ø5
X  : C3
PC : Ø213
SP : Ø1FA
PR : 10100100
     NV BDIZC 2ØØØ
2ØØØ 3C 2ØØ
Ø2ØØ Ø4 P
Ø213 DØ R  39
Ø2Ø6 8C R  4Ø
Ø2Ø9 68 R  41
Ø2ØA 48 R  42
Ø2ØB 18 R  43
Ø2ØC 6D R  44
Ø2ØF 48 R  45
Ø21Ø C8 R  46
Ø211 CØ R  47
Ø213 DØ L  48
ACC: ØF
Y  : Ø6
X  : C3
PC : Ø213
SP : Ø1F9
PR : 10100100
     NV BDIZC 2ØØ
Ø2ØØ Ø5 P
Ø213 DØ R  48
Ø2Ø6 8C R  49
Ø2Ø9 68 R  50
Ø2ØA 48 R  51
Ø2ØB 18 R  52
Ø2ØC 6D R  53
Ø2ØF 48 R  54
Ø21Ø C8 R  55
Ø211 CØ R  56
Ø213 DØ L  57
ACC: 15
Y  : Ø7
```

⑤
```
X  : C3
PC : Ø213
SP : Ø1F8
PR : 10100100
     NV BDIZC 2ØØ
Ø2ØØ Ø6 P
Ø213 DØ R  57
Ø2Ø6 8C R  58
Ø2Ø9 68 R  59
Ø2ØA 48 R  6Ø
Ø2ØB 18 R  61
Ø2ØC 6D R  62
Ø2ØF 48 R  63
Ø21Ø C8 R  64
Ø211 CØ R  65
Ø213 DØ L  66
ACC: 1C
Y  : Ø8
X  : C3
PC : Ø213
SP : Ø1F7
PR : 10100100
     NV BDIZC 2ØØ
Ø2ØØ Ø7 P
Ø213 DØ R  66
Ø2Ø6 8C R  67
Ø2Ø9 68 R  68
Ø2ØA 48 R  69
Ø2ØB 18 R  7Ø
Ø2ØC 6D R  71
Ø2ØF 48 R  72
Ø21Ø C8 R  73
Ø211 CØ R  74
Ø213 DØ L  75
ACC: 24
Y  : Ø9
X  : C3
PC : Ø213
SP : Ø1F6
PR : 10100100
     NV BDIZC 2ØØ
Ø2ØØ Ø8 P
```

⑥
```
Ø213 DØ R  75
Ø2Ø6 8C R  76
Ø2Ø9 68 R  77
Ø2ØA 48 R  78
Ø2ØB 18 R  79
Ø2ØC 6D R  80
Ø2ØF 48 R  81
Ø21Ø C8 R  82
Ø211 CØ R  83
Ø213 DØ L  84
ACC: 2D
Y  : ØA
X  : C3
PC : Ø213
SP : Ø1F5
PR : 10100100
     NV BDIZC 2ØØ
Ø2ØØ Ø9 P
Ø213 DØ R  84
Ø2Ø6 8C R  85
Ø2Ø9 68 R  86
Ø2ØA 48 R  87
Ø2ØB 18 R  88
Ø2ØC 6D R  89
Ø2ØF 48 R  9Ø
Ø21Ø C8 R  91
Ø211 CØ R  92
Ø213 DØ L  93
ACC: 37
Y  : ØB
X  : C3
PC : Ø213
SP : Ø1F4
PR : 00100111
     NV BDIZC 2ØØ
Ø2ØØ ØA P
Ø213 DØ R 93
Ø215 ØØ R 94
Ø217 3C R 95
Ø21A 3C R
Ø21D 3C R
Ø22Ø 47 R
```

⑦
```
Ø222 C3 L
ACC: 56
Y  : ØB
X  : C3
PC : Ø222
SP : Ø1F4
PR : 00100101
     NV BDIZC 1FF
Ø1FF ØØ -
Ø1FE Ø1 -
Ø1FD Ø3 -
Ø1FC Ø6 -
Ø1FB ØA -
Ø1FA ØF -
Ø1F9 15 -
Ø1F8 1C -
Ø1F7 24 -
Ø1F6 2D -
Ø1F5 37 -
Ø1F4 1Ø -
Ø1F3 89 -
Ø1F2 12 -
- - - - - - - - - -
Ø1F1 Ø7 1A7A
1A7A CF .
1A7B 14 1A.
1A7C 5F +
1A7D 1Ø +
1A7E CF .
1A7F 14 1A.
1A8Ø 8Ø 1AØØ
1AØØ ØØ D8.
1AØ1 ØØ 4C.
1AØ2 ØØ CF.
1AØ3 ØØ 14.
1AØ4 ØØ 217
Ø217 3C F8.
Ø218 2E 4C.
Ø219 3C FF.
Ø21A 3C 1.
Ø21B 3C 217
Ø217 F8 R 95
Ø218 4C L 96
```

⑧
```
ACC: 56
Y  : ØB
X  : C3
PC : Ø218
SP : Ø1F4
PR : 00101101
     NV BDIZC P
Ø218 4C R 96
Ø1FF ØØ R 97
Ø2Ø1 AØ L 1
ACC: 56
Y  : ØB
X  : C3
PC : Ø2Ø1
SP : Ø1F4
PR : 00111101
     NV BDIZC 2ØØ
Ø2ØØ ØA P
Ø2Ø1 AØ R 1
Ø2Ø3 A9 R 2
Ø2Ø5 48 R 3
Ø2Ø6 8C R 4
Ø2Ø9 68 R 5
Ø2ØA 48 R 6
Ø2ØB 18 R 7
Ø2ØC 6D R 8
Ø2ØF 48 R 9
Ø21Ø C8 R 10
Ø211 CØ R 11
Ø213 DØ L 12
ACC: Ø1
Y  : Ø2
X  : C3
PC : Ø213
SP : Ø1F2
PR : 10101100
     NV BDIZC 2ØØ
Ø2ØØ Ø1 P
Ø213 DØ R 12
Ø2Ø6 8C R 13
Ø2Ø9 68 R 14
Ø2ØA 48 R 15
```

⑨
```
Ø2ØB 18 R  16
Ø2ØC 6D R  17
Ø2ØF 48 R  18
Ø21Ø C8 R  19
Ø211 CØ R  2Ø
Ø213 DØ L  21
ACC: Ø3
Y  : Ø3
X  : C3
PC : Ø213
SP : Ø1F1
PR : 10101100
     NV BDIZC 2ØØ
Ø2ØØ Ø2 P
Ø213 DØ R  21
Ø2Ø6 8C R  22
Ø2Ø9 68 R  23
Ø2ØA 48 R  24
Ø2ØB 18 R  25
Ø2ØC 6D R  26
Ø2ØF 48 R  27
Ø21Ø C8 R  28
Ø211 CØ R  29
Ø213 DØ L  30
ACC: Ø6
Y  : Ø4
X  : C3
PC : Ø213
SP : Ø1FØ
PR : 10101100
     NV BDIZC 2ØØ
Ø2ØØ Ø3 P
Ø213 DØ R  30
Ø2Ø6 8C R  31
Ø2Ø9 68 R  32
Ø2ØA 48 R  33
Ø2ØB 18 R  34
Ø2ØC 6D R  35
Ø2ØF 48 R  36
Ø21Ø C8 R  37
Ø211 CØ R  38
Ø213 DØ L  39
```

```
⑪
PR : 10101100
     NV BDIZC 200
0200 06 P
0213 D0 R 57
0206 8C R 58
0209 68 R 59
020A 48 R 6Ø
020B 18 R 61
020C 6D R 62
020F 48 R 63
0210 C8 R 64
0211 C0 R 65
0213 D0 L 66
ACC: 28
Y  : 08
X  : C3
PC : 0213
SP : 01EC
PR : 10101100
     NV BDIZC 200
0200 07 P
0213 D0 R 66
0206 8C R 67
0209 68 R 68
020A 48 R 69
020B 18 R 70
020C 6D R 71
020F 48 R 72
0210 C8 R 73
0211 C0 R 74
0213 D0 L 75
ACC: 36
Y  : 09
X  : C3
PC : 0213
SP : 01EB
PR : 10101100
     NV BDIZC 200
0200 08 P
0213 D0 R 75
0206 8C R 76
0209 68 R 77
```

```
⑩
ACC: 10
Y  : 05
X  : C3
PC : 0213
SP : 01EF
PR : 10101100
     NV BDIZC 200
0200 04 P
0213 D0 R 39
0206 8C R 4Ø
0209 68 R 41
020A 48 R 42
020B 18 R 43
020C 6D R 44
020F 48 R 45
0210 C8 R 46
0211 C0 R 47
0213 D0 L 48
ACC: 15
Y  : 06
X  : C3
PC : 0213
SP : 01EE
PR : 10101100
     NV BDIZC 200
0200 05 P
0213 D0 R 48
0206 8C R 49
0209 68 R 5Ø
020A 48 R 51
020B 18 R 52
020C 6D R 53
020F 48 R 54
0210 C8 R 55
0211 C0 R 56
0213 D0 L 57
ACC: 21
Y  : 07
X  : C3
PC : 0213
SP : 01ED
```

```
⑫
020A 48 R 78
020B 18 R 79
020C 6D R 80
020F 48 R 81
0210 C8 R 82
0211 C0 R 83
0213 D0 R 84
0206 8C L 85
ACC: 45
Y  : 0A
X  : C3
PC : 0206
SP : 01EA
PR : 10101100
     NV BDIZC 200
0200 09 P
0206 8C R 85
0209 68 R 86
020A 48 R 87
020B 18 R 88
020C 6D R 89
020F 48 R 9Ø
0210 C8 R 91
0211 C0 R 92
0213 D0 L 93
ACC: 55
Y  : 0B
X  : C3
PC : 0213
SP : 01E9
PR : 00101111
     NV BDIZC 200
0200 0A P
0213 D0 R 93
0215 00 L 94
ACC: 55
Y  : 0B
X  : C3
PC : 0215
SP : 01E9
PR : 00101111
     NV BDIZC 200
```

```
⑬
0200 0A 1FF
01FF 00 -
01FE 01 -
01FD 03 -
01FC 06 -
01FB 0A -
01FA 0F -
01F9 15 -
01F8 1C -
01F7 24 -
01F6 2D -
01F5 37 -
01F4 00 -
01F3 01 -
01F2 03 -
01F1 06 -
01F0 10 -
01EF 15 -
01EE 21 -
01ED 28 -
01EC 36 -
01EB 45 -
01EA 55 -
01E9 10
```

Intermediate phases concerning the 6502 registers were also noted (key L) together with location TEMPY (keys 2 Ø Ø Ø and SP), at points 1, 12, 23, 30, 39, 48, 57, 66, 75, 84 and 93: each of these 'stations' feature a BNE instruction — except for point 23, which was caused by inadvertently pressing the L key. In the end, we are presented with a panorama of all the locations in the stack containing the series of numbers required. Which is what the program was all about.

By the way, table 2 is shown in three columns per page to save space.

At this stage we are not particularly interested in the software developments in table 2, but readers are, of course, welcome to examine the table in greater detail if they so wish. What is important here is the fact that the PM program in the Junior Computer helps to keep the user completely up to date with regard to everything going on inside the machine (by using keys L and P).

Decimal arithmetic using the NUMBERS program

Let us repeat the program, but this time to make a decimal calculation. Exactly the same numbers are involved here, only now they will be decimal numbers as opposed to hexadecimal numbers. In the first instance the numbers were expressed with the characters Ø . . . 9 and A . . . F, in the second only the numbers Ø . . 9 will be required.

As indicated in Appendix 2, jumping to the SAVE routine of original monitor program may cause difficulties during a decimal operation, that is to say, if the last instruction in the sequence was SED as opposed to CLD. Various problems can also be expected when returning to the original monitor during the STEP mode while still operating in decimal. The instructions ADC and SBC appear at a number of places within the PM program and the execution of these instructions in decimal can lead to very strange results. Furthermore, data containing numbers A . . . F just do not occur in decimal arithmetic. The PM software will be considered in much greater detail regarding this aspect in Book 4.

Similar to the process given in Appendix 2 for the original monitor routine, the PM program, or to be more precise, the STEP routine, can be extended by using some of the PIA RAM. This restores the binary situation before the program actually jumps to the STEP routine (see figure 16b). The main requirement for the step procedure is that the NMI vector is pointing to location 1AØØ (the start address of BINAR). This is also necessary if the processor is to return to the main PM routine by way of the ST key. At the same time, the IRQ jump vector must also be pointing to the start of BINAR in case there are any BRK instructions in the user program. Thus, if the D flag is set (decimal arithmetic) just prior to the jump to the PM program (= jump to BINAR) this extra little subroutine is absolutely essential. However, it is rather superfluous in the case of a binary (hexadecimal) calculation – although no harm is done by including it.

The module

In chapter 10 we described the hardware required on the main board of the Junior Computer to disable the step function during the original monitor and the PM routines. As the select line connections on the actual module (EPS 81033-3) are rather confusing, we will recap on the matter once again:

1. **Select line K7 must always be connected** even if the RES, NMI and IRQ vectors are **included** in EPROM on a bus board memory card. This is because the original EPROM contains two indirect jump instructions which must be executed before an interrupt can be carried out (which will have to be done in the step mode, as the effective IRQ and NMI jump vectors are defined in page 1A in the PIA RAM).
2. Select line K6 will be the second link if decimal calculations are to be carried out without using the PM program, in other words, with the aid of the original monitor (see Appendix 2).
3. Select line K4 will be the second link if the PM program is to be used to step through programs, as in the case of NUMBERS in the first section of table 2.

Well, now we can kill two birds with one stone by stepping through the NUMBERS program in the decimal mode! Because of the BINAR routine in figure 16b, a third link, select line K6, will have to be added to the module. How this is done is shown in figure 17. Admittedly, using a germanium diode in series with the K6 select line is not an ideal solution from a technical point of view, but the main thing is: it works (a better solution to this particular problem is described in Book 4). This provision must be made to allow NUMBERS to be stepped through in the decimal mode (second section of table 2). Incidentally, we do not intend that the Junior Computer be switched off and K6 connected, as we are assuming that this was carried out before the NUMBERS program was run in binary!

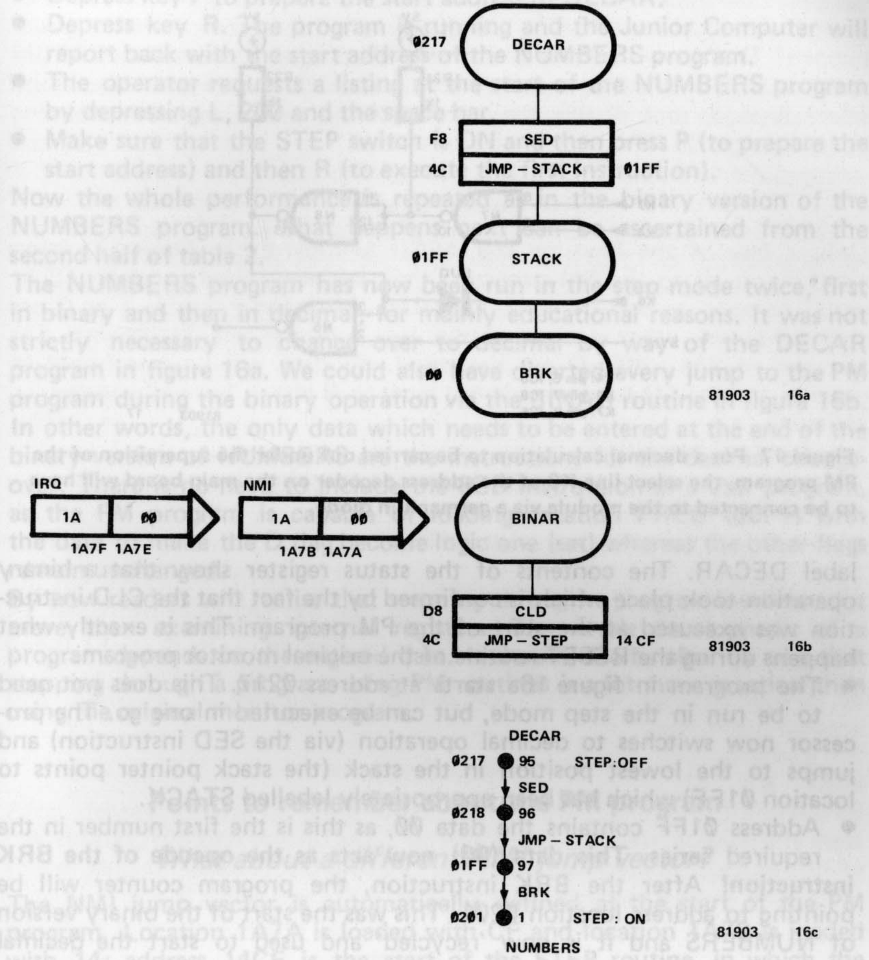


Figure 16. The DECAR routine (16a) enables the NUMBERS program to be run again, this time in the decimal mode. The PM program will therefore have to be extended by using some PIA RAM (16b). Also, the IRQ and NMI jump vectors have to be altered. As a result, the 'railway journey' in figure 15 becomes slightly extended as well (16c).

The NUMBERS program is executed in decimal as follows:

- At the end of the binary version of numbers, the program counter will have reached address Ø217, due to the BRK instruction, which has the

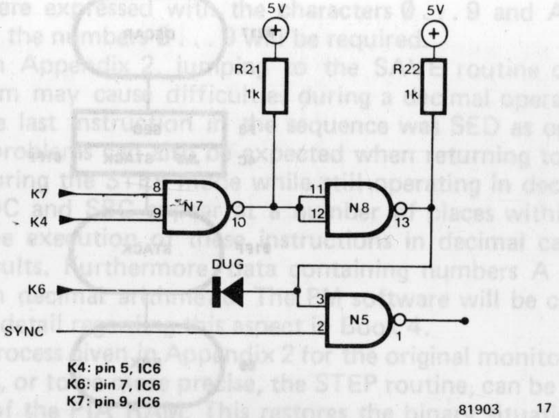


Figure 17. For a decimal calculation to be carried out under the supervision of the PM program, the select line K6 of the address decoder on the main board will have to be connected to the module via a germanium diode.

label DECAR. The contents of the status register show that a binary operation took place which is confirmed by the fact that the CLD instruction was executed at the start of the PM program. This is exactly what happens during the RESET routine of the original monitor program.

- The program in figure 16a starts at address Ø217. This does not need to be run in the step mode, but can be executed in one go. The processor now switches to decimal operation (via the SED instruction) and jumps to the lowest position in the stack (the stack pointer points to location Ø1FF) which has been appropriately labelled STACK.
- Address Ø1FF contains the data ØØ, as this is the first number in the required series. **This data (ØØ) now acts as the opcode of the BRK instruction!** After the BRK instruction, the program counter will be pointing to address location Ø2Ø1. This was the start of the binary version of NUMBERS and it is now 'recycled' and used to start the decimal version.
- The jump from DECAR to NUMBERS by way of the BRK instruction at Ø1FF (see figure 16c) allows the computer to work in decimal instead of binary, without having to modify the program or the start address.

How is this put into practice?

- Assuming that the binary version of the NUMBERS program has just been completed, the program counter will contain the address Ø217 (this can be checked by depressing the P key). At the start of the second section of table 2 a survey of the contents of the internal 6502 registers, that of TEMPY and of the stack locations containing the numbers in the series has been given.
- Type in the data according to figures 16a and 16b. Do not forget to modify the NMI and IRQ jump vectors.
- Turn the STEP switch OFF (not absolutely necessary, but it will take us straight to the start address of NUMBERS). In table 2 this section was in fact run in the STEP mode.

- Depress key P to prepare the start address of DECAR.
- Depress key R. The program is running and the Junior Computer will report back with the start address of the NUMBERS program.
- The operator requests a listing at the start of the NUMBERS program by depressing L, 2ØØ and the space bar.
- Make sure that the STEP switch is ON and then press P (to prepare the start address) and then R (to execute the first instruction).

Now the whole performance is repeated as in the binary version of the NUMBERS program. What happens next can be ascertained from the second half of table 2.

The NUMBERS program has now been run in the step mode twice, first in binary and then in decimal, for mainly educational reasons. It was not strictly necessary to change over to decimal by way of the DECAR program in figure 16a. We could also have diverted every jump to the PM program during the binary operation via the BINAR routine in figure 16b. In other words, the only data which needs to be entered at the end of the binary version of NUMBERS are the instructions for the decimal changeover. There is no need to include the SED instruction in a user program, as the PM program is capable of loading location PREG (ØØF1) with the data to make the D flag become logic one (set) whereas the other flags remain unchanged.

By now readers will realise that stepping through a program involves a lot more than examining internal registers. The whole development of a program depends on the entered data. It should also be clear by now that stepping through a program using PM routines is a lot more practical than using the original monitor program.

Points to remember about the PM program

What about a different NMI jump vector?

The NMI jump vector is automatically defined at the start of the PM program. Location 1A7A is loaded with CF and location 1A7B is loaded with 14; address 14CF is the start of the STEP routine, in which the contents of the CPU registers are stored away and the contents of the program counter is printed on the screen, this being the next start address. The user may, however, change his/her mind about the NMI vector. The operator may like to implement the ST key on the hexadecimal keyboard of the Junior Computer. (Do not forget that each time the ST key is depressed, three stack locations are filled). It is possible to make the NMI vector point to any start address which the user may decide upon. This will have to be carried out after the start of the PM program, as otherwise (if done prematurely) the data at address locations 1A7A and 1A7B will be overwritten by the PM program! The IRQ jump vector can be entered at any time.

Caution!

If either of the keys S or G are depressed (that is to say, a subroutine belonging to the TM program is called) and the STEP switch is ON, no data will be transferred to or from the cassette. The operator can wait until Doomsday for 'READY' to appear on the screen . . . nothing will happen! The reason for this is that the TM program is stored in EPROM and must be enabled by select lines K2 and K3 (see chapter 10). These select lines do not disable the step operation within the corresponding address range. So OFF with the STEP switch!

Error messages

Only a few of the keys on the ASCII keyboard (see figure 12) mean anything at all to the PM program: they are recognised by their ASCII code. As soon as one of the (parameter) command keys M, S or G are depressed, any other perfectly valid key function will be ignored. For instance, if S11, 2ØØ, R are depressed, no data will be transferred to cassette and the PM program will not react to R either, but will answer:

JUNIOR

followed by a blank line.

There are other situations which lead to incomprehension:

a. 'WHAT?'

This desperate cry (sometimes together with JUNIOR) indicates that the PM program does not recognise the key that has just been depressed. In other words, it was not an alphanumeric key (Ø . . . 9, A . . . F) nor a (relevant) function key. If one of the parameter keys are operated (M, S or G) the message 'WHAT?' will always be followed by 'JUNIOR' and another blank line.

For example:

Ø4Ø7   XX   3F ,
WHAT?

Here the data 3F was to be entered into address location Ø4Ø7, but instead of hitting the full stop key we typed in a comma. The comma only means something when it is used in conjunction with keys M and S.

b. 'JUNIOR'

This message has been mentioned on a number of occasions already. In fact, it is output whenever:

1. The PM program is fully activated: both start sequences have been accomplished.
2. When the BREAK key is depressed during the transmission of an ASCII character by the Junior Computer: provided the BRK jump vector is defined as:

1A7C: data FF;   1A7D: data 1Ø.

This occurs automatically after the start of the PM program, but can be modified by the user program.

3. After a hex dump has been printed, by depressing the M key and setting the required parameters. In each case, the computer is reacting quite normally, even though the BREAK key is often used in an emergency. Sometimes, however, the text 'JUNIOR' is displayed because something has gone wrong!

4. After operating the M key and entering an end address which is lower than the start address. Although the parameters are technically correct (no typing mistakes) the error message will be displayed.
5. When the S key is operated and a file number (ID) of ØØ or FF is entered. As you (and the PM program) know, a data block can not possess either number.
6. When a error was made during the entry of a function which requires certain parameters. Only the alphanumeric keys Ø . . . 9 and A . . . F, the ',' key and CR are relevant (the comma is not required if the G key is operated). If irrelevant keys are depressed or if CR, for instance is depressed too soon (before the ',' key), an error will be reported.

The only valid keys are:

M → **HEXDUMP**: (data), (data) CR
S (data), (data), (data) CR
G (data ≠ FF) CR
GFF → **SA**: (data) CR

In every other case an error will be reported. Even, for instance, if the space bar is depressed for more clarity on the video screen.

N.B. 1. Occasionally an error report during the preparation of parameters for a particular function is preceded by the message 'WHAT?'. This aspect will be dealt with during the discussion of the PM software in Book 4.

N.B. 2 What action is taken after an error report? If a key function requiring certain parameters (M, S or G) was involved, start again. So press M, S or G once more and re-enter the parameters. Normally, in most other instances, the work address has to be re-entered

A word on keys S and G

After depressing one of these keys, entering the parameters followed by CR, the cursor will move back to the beginning of the line. The first character in this line will then flash until the text READY appears on the screen. If one of the keys S or G is operated at the start of line, that letter will flash after the CR and before 'READY' appears.

An 'appetiser'

A glimpse behind the cover of Book 4

Which subroutines are incorporated in the PM program? How can these subroutines be implemented in user programs? Such questions will be answered at length in Book 4. However, it is only fair that eager programmers should have something to whet their appetites before Book 4 is available. So here are four examples of ways in which to use the PM program, and which will give you a good idea of what the main software menu has in store for you. At this point in time, the meal is not quite ready, but there is no reason why the kitchen door could not be opened to let the aroma of sizzling bytes waft out . . .

1. Printing data

We have already seen how neatly the PM program prints out a hex dump. How about printing a user program in a similar manner? What about using the two versions of the NUMBERS program (figure 13) as an example? The two series of numbers could be printed out in neat columns.

This can be done with the aid of the program given in figure 18. This is called PRNUMB and makes use of three different PM subroutines:

- **CRLF:** Start at the beginning of a new line (11E8).
- **PRSP:** Print a space (11F3).
- **PRBYT:** Transmit the contents of the accumulator in the form of two ASCII characters (128F).

Let us look at the structure of the PRNUMB program. It starts very much like the NUMBERS program (see figure 13). The first BNE instruction (in the section of program after the ADD label) ensures that this particular section is run until all eleven numbers in the series have been calculated and stored on the stack. Once this has happened, the status of the D flag is examined. The procedure is as follows: the contents of the status register are copied into the accumulator. This taken care of by the instructions PHP and PLA. By means of the instruction AND #Ø8, all the bits in the accumulator (not those in the status register!) are made logic zero, except for the bit corresponding to the D flag. The status of the latter will let us know whether the program section starting at label ADD has been run through once (binary/hexadecimal arithmetic) or twice (decimal arithmetic). For when the bit corresponding to the D flag is logic zero, the program switches to decimal calculation (SED) and then jumps back to PRNUMB.

Readers may well wonder how we can be sure that the first calculation is carried out in binary. Well, the program has to keyed in or read in from cassette using the original monitor program or, preferably, the PM and TM programs. After the start of either the original monitor or the PM program, the Junior Computer will automatically be in the binary mode.

When the D flag is logic one, to show that both series of numbers have been calculated, the program jumps to the label READY. The second section of PRNUMB prints out the numbers in a clear and orderly fashion. Now you may wonder why this was not done during the ADD routine in the first place. The problem is, this can be done during the binary calculation, but not during the decimal version. This is because the PM subroutines are likely to go haywire if they are set to work via a JSR instruction while the computer is still in the decimal mode. Although the jump to the STEP routine of the PM program by way of the BINAR routine and while the D flag is still set is quite plausible, problems are bound to arise if the binary adjustment is omitted and the PM program is entered by way of one of its subroutines. For this reason the READY label is organised at once: CLD. Next, the jump to subroutine CRLF brings us to a fresh line on the display.

The purpose of PRNUMB is to print out the two series of numbers in two rows: first the binary row and below it the decimal row. For this the Y index register is used. The contents of the Y register are made equal to 15 and the accumulator is loaded using post indexed indirect addressing method, the operand address being Ø1EA (STACK). By looking at the

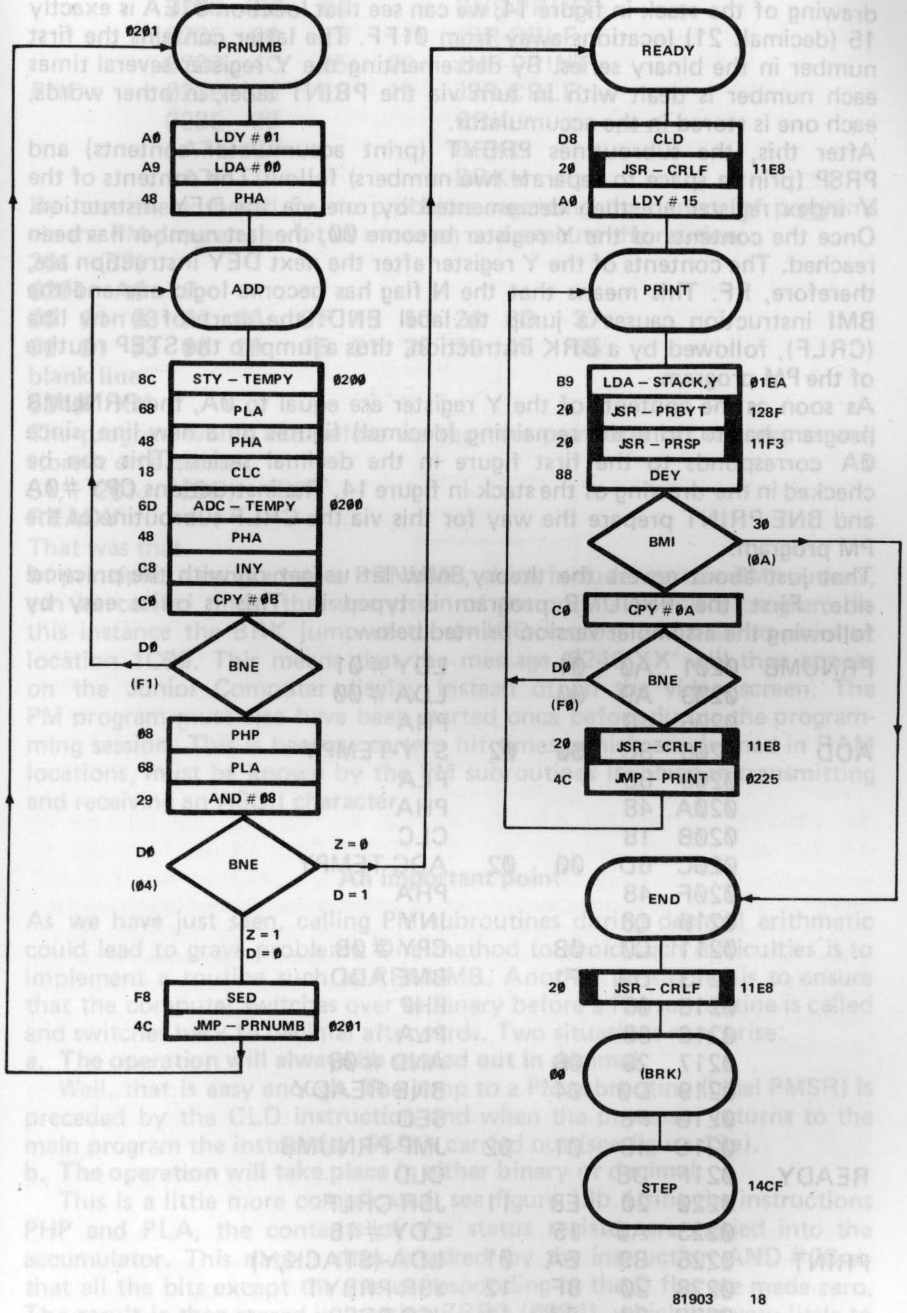


Figure 18. The PRNUMB program enables the series of numbers (see figure 13) to be calculated in both binary and decimal. The contents of the stack (see figure 14) are printed as a row of hexadecimal numbers followed by a row of decimal numbers.

drawing of the stack in figure 14, we can see that location Ø1EA is exactly 15 (decimal: 21) locations away from Ø1FF. The latter contains the first number in the binary series. By decrementing the Y register several times each number is dealt with in turn via the PRINT label, in other words, each one is stored in the accumulator.

After this, the subroutines PRBYT (print accumulator contents) and PRSP (print a space to separate two numbers) follow. The contents of the Y index register are then decremented by one via the DEY instruction. Once the contents of the Y register become ØØ, the last number has been reached. The contents of the Y register after the next DEY instruction are, therefore, FF. This means that the N flag has become logic one and the BMI instruction causes a jump to label END: the start of a new line (CRLF), followed by a BRK instruction, thus a jump to the STEP routine of the PM program.

As soon as the contents of the Y register are equal to ØA, the PRNUMB program has to print the remaining (decimal) figures on a new line, since ØA corresponds to the first figure in the decimal series. This can be checked in the drawing of the stack in figure 14. The instruction CPY #ØA and BNE PRINT prepare the way for this via the CRLF subroutine of the PM program.

That just about covers the theory, now let us get on with the practical side. First the PRNUMB program is typed in. This is quite easy by following the assembler version printed below:

```
PRNUMB  Ø2Ø1  AØ  Ø1       LDY #Ø1
        Ø2Ø3  A9  ØØ       LDA #ØØ
        Ø2Ø5  48           PHA
ADD     Ø2Ø6  8C  ØØ  Ø2   STY-TEMPY
        Ø2Ø9  68           PLA
        Ø2ØA  48           PHA
        Ø2ØB  18           CLC
        Ø2ØC  6D  ØØ  Ø2   ADC-TEMPY
        Ø2ØF  48           PHA
        Ø21Ø  C8           INY
        Ø211  CØ  ØB       CPY #ØB
        Ø213  DØ  F1       BNE ADD
        Ø215  Ø8           PHP
        Ø216  68           PLA
        Ø217  29  ØB       AND #Ø8
        Ø219  DØ  Ø4       BNE READY
        Ø21B  F8           SED
        Ø21C  4C  Ø1  Ø2   JMP-PRNUMB
READY   Ø21F  D8           CLD
        Ø22Ø  2Ø  E8  11   JSR-CRLF
        Ø223  AØ  15       LDY #15
PRINT   Ø225  B9  EA  Ø1   LDA-(STACK,Y)
        Ø228  2Ø  8F  12   JSR-PRBYT
        Ø22B  2Ø  F3  11   JSR-PRSP
        Ø22E  88           DEY
        Ø22F  3Ø  ØA       BMI END
        Ø231  CØ  ØA       CPY #ØA
        Ø233  DØ  FØ       BNE PRINT
        Ø235  2Ø  E8  11   JSR-CRLF
        Ø238  4C  25  Ø2   JMP-PRINT
END     Ø23B  2Ø  E8  11   JSR-CRLF
        Ø23E  ØØ           BRK
        1A7E  CF           BRKL
        1A7F  14           BRKH
```

By now there should be no problems regarding the entry of programs via the PM program, so let us move on and execute the routine:

```
2Ø1  (SP)
Ø2Ø1  AØ  R
ØØ  Ø1  Ø3  Ø6  ØA  ØF  15  1C  24  2D  37
ØØ  Ø1  Ø3  Ø6  1Ø  15  21  28  36  45  55
blank line
Ø24Ø  XX
```

The program works and before we turn the computer off we may as well store it on cassette:

```
S1,  2Ø1,  23F,  (CR)
READY
```

That was that.

In principle, a program like PRNUMB which includes certain PM routines, can be carried under the supervision of the original monitor program. In this instance the BRK jump vector (= IRQ vector) will have to point at location 1CØØ. This means that the message 'Ø24Ø XX' will then appear on the Junior Computer display instead of on the video screen. The PM program must also have been started once before during the programming session. This is because certain bit times, which are defined in RAM locations, must be known by the PM subroutines involved in transmitting and receiving an ASCII character.

An important point

As we have just seen, calling PM subroutines during decimal arithmetic could lead to grave problems. One method to avoid such difficulties is to implement a routine such as PRNUMB. Another possibility is to ensure that the computer switches over to binary before a PM subroutine is called and switches back to decimal afterwards. Two situations may arise:

a. The operation will always be carried out in decimal.

Well, that is easy enough. The jump to a PM subroutine (label PMSR) is preceded by the CLD instruction and when the processor returns to the main program the instruction SED is carried out (see figure 19a).

b. The operation will take place in either binary or decimal.

This is a little more complicated, see figure 19b. Using the instructions PHP and PLA, the contents of the status register are copied into the accumulator. This data is then 'masked' by the instruction AND #Ø8 so that all the bits except the one corresponding to the D flag are made zero. The result is then stored in location PREG (ØØE1), which has very little to do seeing as the original monitor program can no longer be stepped through.

The instruction CLD follows, which is not strictly necessary, and then

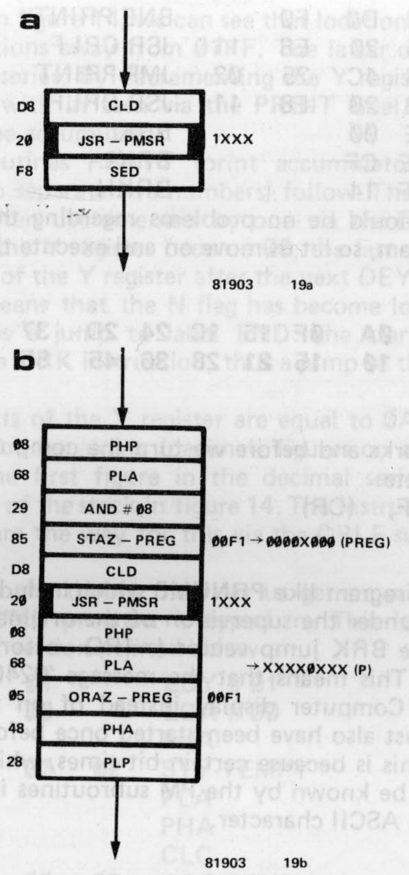


Figure 19. The PM subroutines should never be called during decimal operations. If they are required, the computer will have to change over to binary arithmetic for a while (19a). If the program involves both binary and decimal calculations (not at the same time!), the original status of the D flag must be restored upon the return from a PM subroutine (19b).

the relevant PM subroutine is called and run. Many subroutines cause one or more of the various flags to be set or reset. This will be elaborated on in Book 4. In other words, the contents of the status register at the end of the subroutine are of vital importance. Only the D flag must be restored to its original value.

This is accomplished by fetching the contents of the status register and storing them in the accumulator (the instructions PHP and PLA). We know that the D flag is reset at the end of the PM subroutine. The D flag is restored to its original value by means of the instruction ORAZ-PREG, which will have no effect on any of the other bits in the accumulator. The result is then stored in the status register. by means of the instructions PHA and PLP.

2. Graphic display

Are you familiar with the PM subroutine PRCHA? Yes and no. Actually, this routine was introduced twice in succession during the PRBYT subroutine, which we learned about in the first example (PRNUMB, see figure 18). The PRCHA routine is devoted to transmitting, or rather printing, the contents of the accumulator. The important thing to know here is that the contents of the X index register are exactly the same after the PRCHA routine as they were before, in spite of the intensive activity during the subroutine. The start address of PRCHA is 1334.

In the first example, alphanumeric data was printed; in the last example we will print a short piece of text, but before this, it is time to have a bit of fun with a graphic display in our second example. Again the programs need not serve any particularly significant purpose: they are just exercises to enable us to improve our programming skills.

The idea is to display a chessboard on the TV screen. Obviously, it will have to be a very simple pattern due to the limitations of the Elekterminal. It will have eight rows of eight squares with the usual pattern of 32 white squares and 32 black squares. A white square is indicated by a blank space and a black square is marked by a cross (X). The squares themselves are not outlined.

The chessboard should be square on the screen. Since there is a space between the rows of characters on the screen (about the same size as the space occupied by a letter), extra spaces will have to be printed next to those representing a white square, to make the graphic display appear even. This is not quite possible, but we can reduce the size of the TV picture. Most sets provide a control for this purpose. As a result, horizontal black bars appear at the top and bottom of the screen. We know what happens next. The question is: how?

Look at the following rows of data very carefully:

ØØ	Ø1	Ø2	Ø3	Ø4	Ø5	Ø6	Ø7	Ø8	Ø9	ØA	ØB	ØC	ØD	ØE	ØF	1Ø
2Ø	2Ø	58	2Ø	2Ø	2Ø	58	2Ø	2Ø	2Ø	58	2Ø	2Ø	2Ø	58	ØD	ØA

The first row contains a series of consecutive numbers and the second row contains ASCII coded data: 2Ø stands for a space, 58 represents a capital 'X', ØD and ØA are the 'invisible' commands for carriage return (CR) and line feed (LF), respectively. The first row contains a number of values allocated to the Y index register one after the other. The second contains data which is stored in the accumulator by means of post indexed indirect addressing and then printed with the aid of the PRCHA subroutine contained in the PM program. The data row corresponds to one line of the chessboard, but not all the data in the row is used to print one chessboard line. The data row mentioned is actually dealt with nine times: once to make sure the chessboard display starts at the beginning of a new line and another eight times to display a row of chessboard squares. The X index register is used to keep an eye on the progress. This is incremented from ØØ to Ø9:

a. When X = ØØ, Y = ØF; a new line is started.
b. When X = Ø1, Ø3, Ø5 and Ø7, Y = ØØ; the chessboard line starts with a white square.
c. When X = Ø2, Ø4, Ø6 and Ø8, Y = Ø2; the chessboard line starts with a black square.

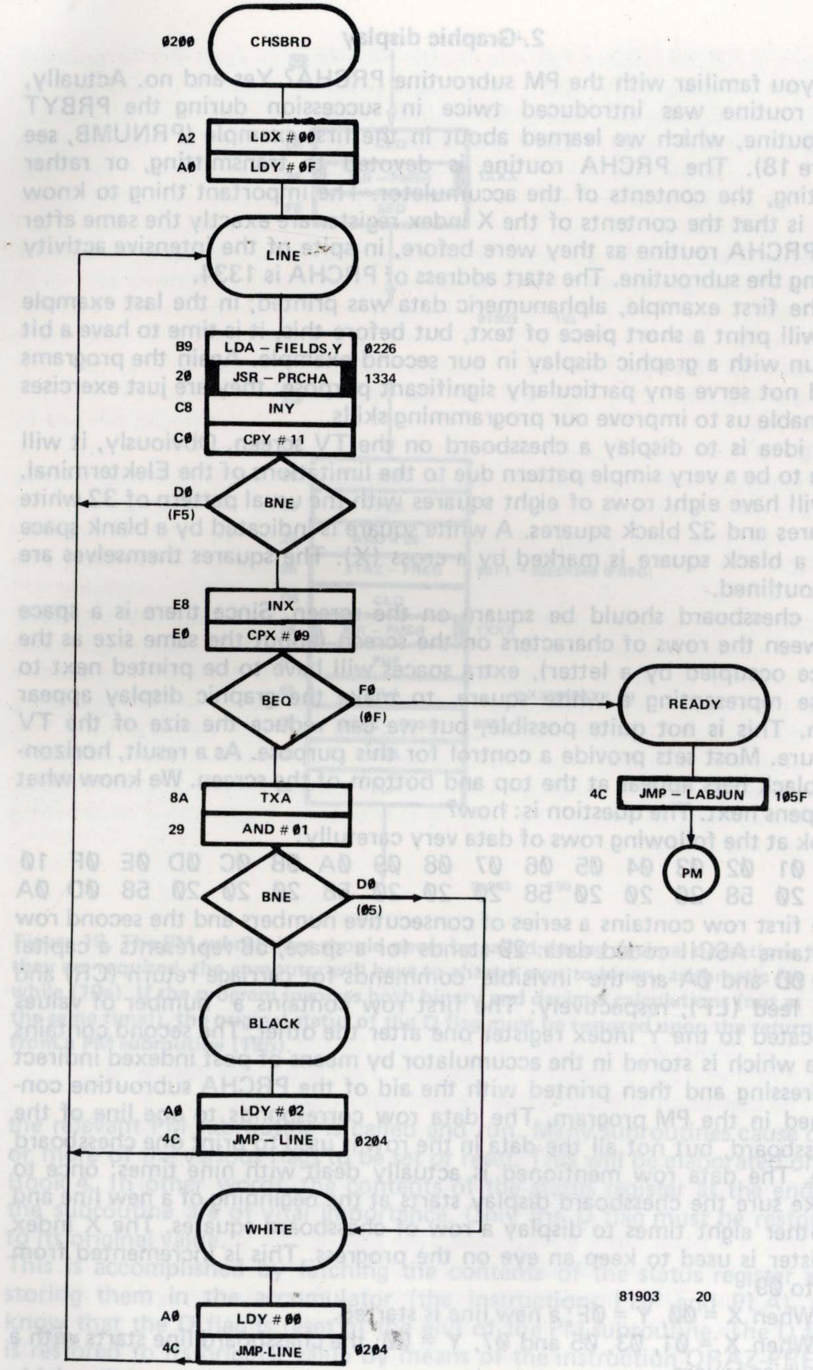


Figure 20. The CHSBRD program displays a simplified chessboard pattern on the TV screen.

d. When the value in the X register has reached Ø9, the computer knows that it has finished printing out the chessboard and will display the message 'JUNIOR'.

That just about covers the main points of the chessboard program CHSBRD, given in figure 20. The section of program between label LINE and the next BNE instruction is run several times in order to print out one line of the chessboard. The data row mentioned earlier can be found in the form of a look-up table under the label FIELDS. The contents of the accumulator which are to be printed are fetched from the relevant table location. Afterwards, the contents of the Y index register are incremented by one. As soon as the contents of the Y register reach 11, the line will be complete and the computer proceeds to the next line. Not surprisingly, the contents of the X index register are then incremented (INX) and the computer checks to see whether the value in the X register has reached Ø9 yet.

If one or more lines are still to be printed, the value contained in the X index register is examined to see whether it is even or odd. Should the next line start with a black square (Y = Ø2) or with a white square (Y = ØØ)? How does the computer know whether the value in the X register is odd or even? This is determined by copying the contents of the X register into the accumulator (TXA) and then masking all the bits in the accumulator except the least significant one (AND # Ø1). This means that if the value in the X register was odd, the Z flag will be reset and if the value was even, the Z flag will be set. After the masking process the BNE instruction acts like a set of points on a railway line and diverts the processor to label BLACK or label WHITE depending on the status of the Z flag.

As soon as the entire chessboard has been printed on the screen (the contents of the X register = Ø9), the program jumps to the READY label in figure 20 and then to the LABJUN label in the PM program. This latter section of the PM program takes care of the 'JUNIOR' report and then jumps to the middle of the PM program to wait for a key to be depressed.

So much for the theory behind the CHSBRD program, now for the practical side. The relevant data is presented in assembler format below:

```
CHSBRD  Ø2ØØ  A2  ØØ        LDX # ØØ
        Ø2Ø2  AØ  ØF        LDY # ØF
LINE    Ø2Ø4  B9  26   Ø2   LDA-FIELDS,Y
        Ø2Ø7  2Ø  34   13   JSR-PRCHA
        Ø2ØA  C8            INY
        Ø2ØB  CØ  11        CPY # 11
        Ø2ØD  DØ  F5        BNE LINE
        Ø2ØF  E8            INX
        Ø21Ø  EØ  Ø9        CPX # Ø9
        Ø212  FØ  ØF        BEQ READY
        Ø214  8A            TXA
        Ø215  29  Ø1        AND # Ø1
        Ø217  DØ  Ø5        BNE WHITE
BLACK   Ø219  AØ  Ø2        LDY # Ø2
        Ø21B  4C  Ø4   Ø2   JMP-LINE
WHITE   Ø21E  AØ  ØØ        LDY # ØØ
        Ø22Ø  4C  Ø4   Ø2   JMP-LINE
```

]

```
READY   Ø223  4C  5F  1Ø   JMP-LABJUN
FIELDS  Ø226  2Ø
        Ø227  2Ø
        Ø228  58
        Ø229  2Ø
        Ø22A  2Ø
        Ø22B  2Ø
        Ø22C  58
        Ø22D  2Ø
        Ø22E  2Ø
        Ø22F  2Ø
        Ø23Ø  58
        Ø231  2Ø
        Ø232  2Ø
        Ø233  2Ø
        Ø234  58
        Ø235  ØD
        Ø236  ØA
```

We only need to type this in once, so:

```
S2, 2ØØ, 237 (CR)
READY
```

The start address of the CHSBRD program is our old favourite, Ø2ØØ. The chessboard pattern is meant to look like this:

```
 X X X X
X X X X
 X X X X
X X X X
 X X X X
X X X X
 X X X X
X X X X
```

but then with a few more spaces. In practice, the result is more like this:

```
      X     X     X     X
   X     X     X     X
      X     X     X     X
   X     X     X     X
      X     X     X     X
   X     X     X     X
      X     X     X     X
   X     X     X     X
```

3. Prepare your own ASCII table!

An ASCII table is part and parcel of any book on computers and this book is no exception: an ASCII table has been printed in Appendix 7. The Elektor article on the ASCII keyboard also contains a full table with all 128 possibilities listed. Of course, it all looks very good on paper, but how about writing a program to print out the ASCII code of a depressed key followed by its actual binary value.

This is quite feasible, as we are about to find out. The previous two examples involved PM subroutines purely for printing purposes. For a program to display the ASCII code of any depressed key, we require a subroutine which waits for an ASCII key to be depressed. Such a subroutine can be found in the PM program by the name of RECCHA.

All this subroutine does is to detect the presence of an ASCII code transmitted by the keyboard to the Junior Computer and the video terminal (or printer). Thus, its name is quite appropriate: RECeive CHAracter. The ASCII code corresponding to the key in question is determined according to whether a normal key function is involved, or whether one which requires simultaneous operation of the shift (SFT) or control (CNTRL) keys. After the RECCHA subroutine the ASCII code of the depressed key will be stored in the accumulator.

The program needed to display the ASCII code of any selected key is called, quite simply, ASCII and is shown in figure 21. The start address of the program is ØØØØ. The program will certainly not use up too much of the computer RAM, as it is only 26 bytes long.

Firstly, the subroutine CRLF sets things going: start at the beginning of a new line. Then the computer waits for the user to depress a key. This is then printed at the beginning of a line on the screen.

The ASCII code of the particular key is then copied from the accumulator into the Y register (TAY). This is because the contents of the accumulator will alter during the following PRSP routine. This latter routine prints a space between the key character and the following hexadecimal representation. The latter occurs by restoring the previous contents of the accumulator (TYA) and calling the PRBYT subroutine. We already know the PRBYT subroutine from our first program example PRNUMB. This makes sure that first the high order nibble and then the low order nibble contained in the accumulator are printed in that order. The value of each

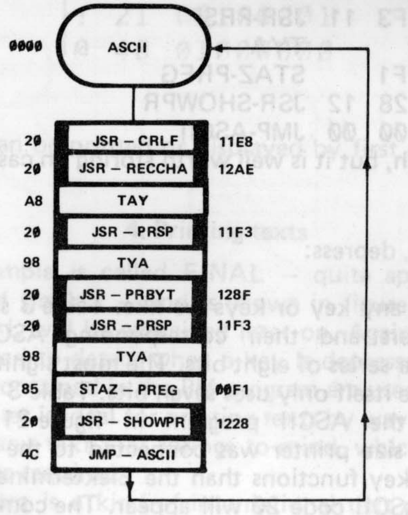


Figure 21. The ASCII program enables the ASCII code and the binary value of any (ASCII) key which is depressed to be displayed on the screen or printed on paper.

nibble can be anywhere between Ø and F, and each is translated into its corresponding ASCII code prior to the actual transmission.

After these two hexadecimal nibbles have been printed, a second space is 'displayed'. This brings us to the section of program which outputs the ASCII code as a series of eight bits. First, the original contents of the accumulator are restored once more (TYA). Then this data is copied into location PREG (ØØF1) and the program jumps to subroutine SHOWPR.

The contents of the accumulator are stored in location PREG for the following reason. During a listing the contents of the status register are printed out bit by bit. This is taken care of by the subroutine SHOWPR. The original contents of PREG are irrelevant here, as this location is being filled with new data.

After all the bits in the ASCII code have been printed, the program returns from the SHOWPR routine and jumps back to the start of the ASCII program. Once a new line has been prepared, the computer waits until a new key is depressed, and so on. The ASCII program is therefore a continuous loop. This can be exited from by depressing the RST key on the hexadecimal keyboard and starting the PM program anew, or by depressing the BREAK key while an ASCII character is being transmitted.

Since the main routine of the PM program is not concerned here, the text 'WHAT?' will not appear on the screen when a key not associated with the PM program is depressed. Therefore, all the ASCII keys on the keyboard can be displayed in this manner.

The ASCII program in figure 21 can be entered very quickly:

```
ASCII  ØØØØ  2Ø  E8  11  JSR-CRLF
       ØØØ3  2Ø  AE  12  JSR-RECCHA
       ØØØ6  A8          TAY
       ØØØ7  2Ø  F3  11  JSR-PRSP
       ØØØA  98          TYA
       ØØØB  2Ø  8F  12  JSR-PRBYT
       ØØØE  2Ø  F3  11  JSR-PRSP
       ØØ11  98          TYA
       ØØ12  85  F1      STAZ-PREG
       ØØ14  2Ø  28  12  JSR-SHOWPR
       ØØ17  4C  ØØ  ØØ  JMP-ASCII
```

It does not look much, but it is well worth storing on cassette, thus:

```
S3, 1A (CR)
READY
```

To start the program, depress:

(SP) R

We can now depress any key or keys we like. Table 3 shows a number of ASCII key characters and their corresponding ASCII code both in hexadecimal and as a series of eight bits. The most significant bit is always zero; the ASCII code itself only uses seven bits. Table 3 was actually made with the aid of the ASCII program in figure 21. Instead of the Elekterminal, a full size printer was connected to the Junior Computer. This provides more key functions than the Elekterminal. When the space bar is pressed, the ASCII code 2Ø will appear. The commands CR and LF give ØD and ØA, respectively.

The ASCII program can be applied in the fourth and last program example, where any texts can be printed or displayed by first entering the relevant ASCII codes.

Table 3. A 'hard copy' ASCII table which was printed with the aid of the ASCII program in figure 21.

Ø	3Ø	ØØ11ØØØØ	P	5Ø	Ø1Ø1ØØØØ	#	23	ØØ1ØØØ11
1	31	ØØ11ØØØ1	Q	51	Ø1Ø1ØØØ1	$	24	ØØ1ØØ1ØØ
2	32	ØØ11ØØ1Ø	R	52	Ø1Ø1ØØ1Ø	%	25	ØØ1ØØ1Ø1
3	33	ØØ11ØØ11	S	53	Ø1Ø1ØØ11	^	5E	Ø1Ø1111Ø
4	34	ØØ11Ø1ØØ	T	54	Ø1Ø1Ø1ØØ	&	26	ØØ1ØØ11Ø
5	35	ØØ11Ø1Ø1	U	55	Ø1Ø1Ø1Ø1	*	2A	ØØ1Ø1Ø1Ø
6	36	ØØ11Ø11Ø	V	56	Ø1Ø1Ø11Ø	(	28	ØØ1Ø1ØØØ
7	37	ØØ11Ø111	W	57	Ø1Ø1Ø111	)	29	ØØ1Ø1ØØ1
8	38	ØØ111ØØØ	X	58	Ø1Ø11ØØØ		5F	Ø1Ø11111
9	39	ØØ111ØØ1	Y	59	Ø1Ø11ØØ1	—	5F	Ø1Ø11111
A	41	Ø1ØØØØØ1	Z	5A	Ø1Ø11Ø1Ø	+	2B	ØØ1Ø1Ø11
B	42	Ø1ØØØØ1Ø	,	2C	ØØ1Ø11ØØ	~	7E	Ø111111Ø
C	43	Ø1ØØØØ11	.	2E	ØØ1Ø111Ø	]	5D	Ø1Ø111Ø1
D	44	Ø1ØØØ1ØØ		2Ø	ØØ1ØØØØØ	|	7C	Ø11111ØØ
E	45	Ø1ØØØ1Ø1	/	2F	ØØ1Ø1111	:	3A	ØØ111Ø1Ø
F	46	Ø1ØØØ11Ø	{	7B	Ø1111Ø11	"	22	ØØ1ØØØ1Ø
G	47	Ø1ØØØ111	'	27	ØØ1ØØ111	}	7D	Ø11111Ø1
H	48	Ø1ØØ1ØØØ	;	3B	ØØ111Ø11	<	3C	ØØ1111ØØ
I	49	Ø1ØØ1ØØ1	\	5C	Ø1Ø111ØØ	>	3E	ØØ11111Ø
J	4A	Ø1ØØ1Ø1Ø	[	5B	Ø1Ø11Ø11	?	3F	ØØ111111
K	4B	Ø1ØØ1Ø11	`	6Ø	Ø11ØØØØØ		ØD	ØØØØ11Ø1
L	4C	Ø1ØØ11ØØ	=	3D	ØØ1111Ø1			
M	4D	Ø1ØØ11Ø1	–	2D	ØØ1Ø11Ø1		ØA	ØØØØ1Ø1Ø
N	4E	Ø1ØØ111Ø	!	21	ØØ1ØØØØ1			
O	4F	Ø1ØØ1111	@	4Ø	Ø1ØØØØØØ			

4. Printing texts

This program example is called FINAL — quite appropriate as we are nearing the end of Book 3 — and is shown in figure 22. The real reason for calling it FINAL will be explained later on. Again, the PM subroutine RECCHA is required to detect when a key is depressed. In addition, two print subroutines contained in the PM program are used.

The Junior Computer is ideal for printing texts by way of the PM program. By this we mean any text that springs to mind, which the computer can print out as often as required.

The FINAL program is a kind of international text routine. The keys N (Dutch), D (German), E (English) and F (French) acquire a special importance, for depressing one of these will make the Junior Computer display a

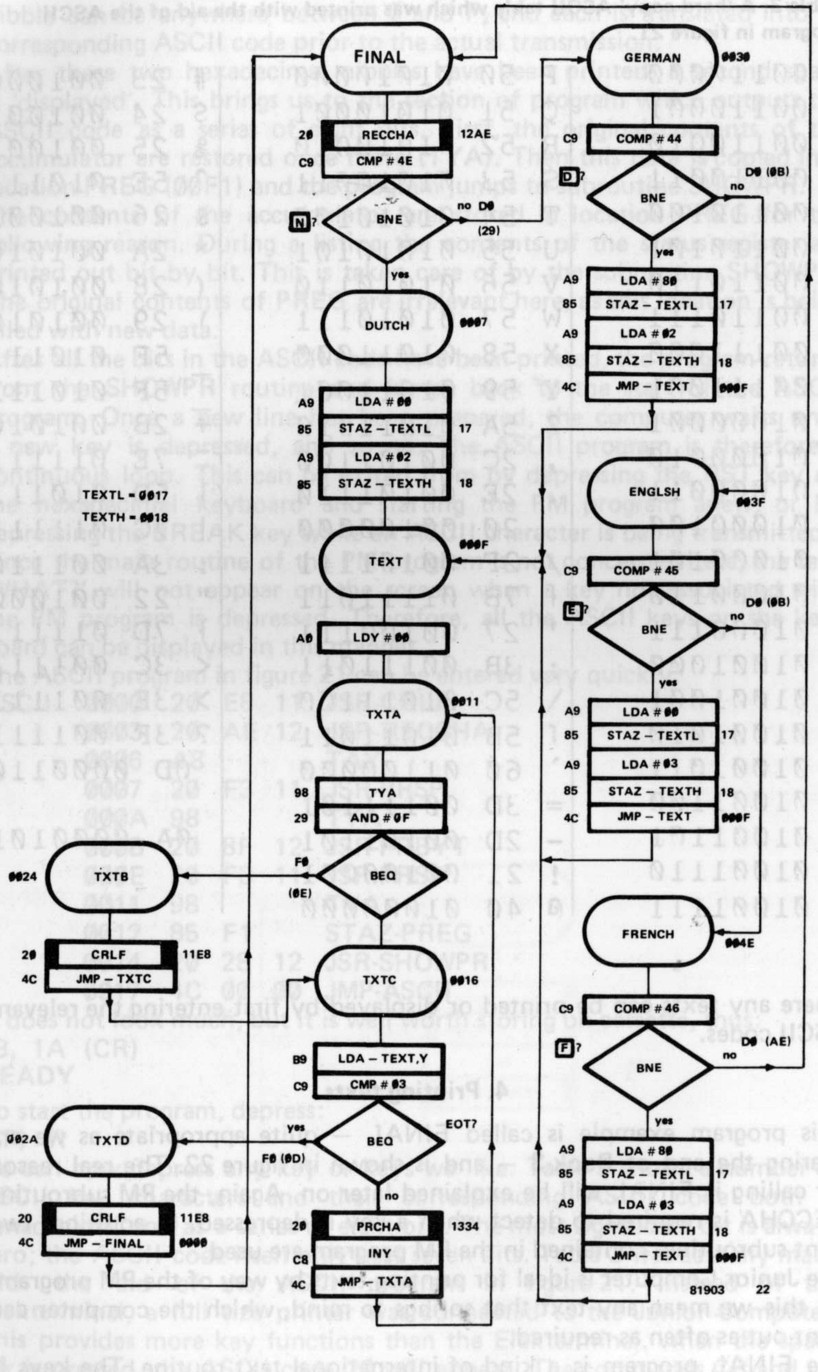


Figure 22. The FINAL program allows the user to display one of four language texts on the screen by depressing the corresponding ASCII key.

short piece of text in one of the four languages. Obviously, the texts need to be entered into RAM beforehand.

Why four languages? Well for one thing, the Junior Computer books are published in four languages. At the same time, it gives the programmer an idea of how to go about assigning different keys to perform different tasks in his/her own software.

The texts are printed in a series of lines, with up to sixteen characters per line. Pages Ø2 and Ø3 are available in the original RAM for storing the texts. Each language is given an equal amount of memory space.

N:	Ø2ØØ . . . Ø27F;	TEXTH = Ø2;	TEXTL = ØØ
D:	Ø28Ø . . . Ø2FF;	TEXTH = Ø2;	TEXTL = 8Ø
E:	Ø3ØØ . . . Ø37F;	TEXTH = Ø3;	TEXTL = ØØ
F:	Ø38Ø . . . Ø3FF;	TEXTH = Ø3;	TEXTL = 8Ø

This means that each text can consist of up to eight lines of sixteen characters. The block of text must end with the End Of Text (EOT) character Ø3, so that each block consists of no more than 127 characters. They must be entered in ASCII format, see Appendix 7.

The actual printing section of the FINAL program starts at the label TEXT. First of all, the contents of the Y index register are made zero. The contents of the Y register are then tested to see whether they are equal to XF, where X = Ø . . . 7. If so, a new line will have to be started, before any more characters can be printed. This is accomplished by copying the contents of the Y register into the accumulator (TYA) and masking the four most significant bits (AND # ØF). The Z flag can then be examined by means of the BEQ instruction.

Next, the accumulator is loaded with the (following) character to be printed. This is accomplished by using post indexed indirect addressing. The memory locations TEXTH and TEXTL are loaded with data which corresponds to the first address of the text memory range in question (depending on which language was chosen).

Each piece of text must end with the EOT character, as otherwise the TV screen will be filled with rubbish. As soon as an EOT character is detected, the subsequent BEQ instruction brings the computer to the beginning of a new line via the CRLF subroutine and then back to the start of FINAL. Until this happens, the contents of the accumulator are printed (PRCHA), the contents of the Y index register are incremented and the next character to be printed is fetched from the text memory.

Now for the key routine. The FINAL program starts with the RECCHA subroutine. When an ASCII character is received, this is stored in the accumulator. Next, a series of comparisons (CMP) and branches (BNE) check whether the character in question belongs to one of the keys N, D, E or F. If not, return to FINAL. If so, adjust the operand of the text memory to the language selected.

Right, now it is time to enter the FINAL program. This is accomplished as follows:

```
FINAL    ØØØØ   2Ø   AE   12    JSR-RECCHA
         ØØØ3   C9   4E         CMP # 4E
         ØØØ5   DØ   29         BNE GERMAN
DUTCH    ØØØ7   A9   ØØ         LDA # ØØ
         ØØØ9   85   17         STAZ-TEXTL
         000B  A9   02          LDA # 02
         000D  85   18          STAZ-TEXTH
TEXT     000F  A0   00          LDY # 00
TXTA     0011  98               TYA
         0012  29   0F          AND # 0F
         0014  F0   0E          BEQ TXTB
TXTC     0016  B9   XX    XX    LDA-TEXT,Y
         0019  C9   03          CMP # 03
         001B  F0   0D          BEQ TXTD
         001D  20   34    13    JSR-PRCHA
         0020  C8               INY
         0021  4C   11    00    JMP-TXTA
TXTB     0024  20   E8    11    JSR-CRLF
         0027  4C   16    00    JMP-TXTC
TXTD     002A  20   E8    11    JSR-CRLF
         002D  4C   00    00    JMP-FINAL
GERMAN   0030  C9   44          CMP # 44
         0032  D0   0B          BNE ENGLSH
         0034  A9   80          LDA # 80
         0036  85   17          STAZ-TEXTL
         0038  A9   02          LDA # 02
         003A  85   18          STAZ-TEXTH
         003C  4C   0F    00    JMP-TEXT
ENGLSH   003F  C9   45          CMP # 45
         0041  D0   0B          BNE FRENCH
         0043  A9   00          LDA # 00
         0045  85   17          STAZ-TEXTL
         0047  A9   03          LDA # 03
         0049  85   18          STAZ-TEXTH
         004B  4C   0F    00    JMP-TEXT
FRENCH   004E  C9   46          CMP # 46
         0050  D0   AE          BNE FINAL
         0052  A9   80          LDA # 80
         0054  85   17          STAZ-TEXTL
         0056  A9   03          LDA # 03
         0058  85   18          STAZ-TEXTH
         005A  4C   0F    00    JMP-TEXT
```

This concludes the entry of the FINAL program. The IRQ jump vector does not need to be defined as there is no BRK instruction in the program. Theoretically, the program is an endless loop. It can, however, be left by depressing the RST key on the hexadecimal keyboard. This will take us back to the original monitor program. The PM program can then be started as follows:

```
1000   GO   RUBOUT
```

Alternatively, the program can be exited from by depressing the BREAK key on the ASCII keyboard while the text is being printed. A third method is to depress the ST key on the hexadecimal keyboard. The BREAK and ST solutions will only work if the correct vectors were defined in accordance with the start routine of the PM program.

Now for the actual text(s). Let us start with the Dutch version:

```
M
HEXDUMP:  200,230
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0200: 42 45 53 54 45 20 4D 45 4E 53 45 4E 21 44 49 54
0210: 57 41 53 20 48 45 54 20 44 41 4E 2E 54 4F 54 20
0220: 5A 49 45 4E 53 20 49 4E 20 42 4F 45 4B 20 34 21
0230: 03

JUNIOR
```

Do not forget the EOT character, 03, at address location 0230, as otherwise random data will be printed after the real text and this is likely to continue for ever (unless there happens to be 03 somewhere). This random data may well contain certain control functions, so that data may suddenly appear at the top of the screen and wipe out the existing text! In other words, the EOT character is very important!
Next the German text:

```
M
HEXDUMP:  280,2BF
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0280: 4C 49 45 42 45 52 20 4C 45 53 45 52 21 44 41 53
0290: 57 41 52 20 45 53 20 46 55 45 52 20 48 45 55 2D
02A0: 54 45 2E 41 55 46 20 57 49 45 44 45 52 53 45 2D
02B0: 48 45 4E 20 49 4E 20 42 55 43 48 20 34 21 21 03
02C0:

JUNIOR
```

The next one should be easy, it is English:

```
M
HEXDUMP:  300,340
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0300: 44 45 41 52 20 4D 52 2F 4D 52 53 2F 4D 49 53 53
0310: 4A 55 4E 49 4F 52 21 49 54 27 53 20 54 49 4D 45
0320: 54 4F 20 47 4F 2E 53 45 45 20 59 4F 55 20 20 20
0330: 41 47 41 49 4E 20 49 4E 20 42 4F 4F 4B 20 34 21
0340: 03

JUNIOR
```

And finally, across the channel, for the French version:

```
M
HEXDUMP:  380,3E0
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0380: 43 48 45 52 53 20 4C 45 43 54 45 55 52 53 2C 20
0390: 43 27 45 53 54 20 46 49 4E 49 20 50 4F 55 52 20
03A0: 41 55 4A 4F 55 52 44 27 48 55 49 21 20 20 20 20
03B0: 52 45 4E 44 45 5A 2D 56 4F 55 53 20 41 55 58 20
03C0: 50 52 45 4D 49 45 52 45 53 20 50 41 47 45 53 20
03D0: 44 55 20 4C 49 56 52 45 20 51 55 41 54 52 45 20
03E0: 03

JUNIOR
```

Now the whole program is complete and can be stored on cassette:

```
S4, , 5D (CR)
READY          (press the pause button)
S5, 200, 231 (CR)
READY          (press the pause button)
S6, 380, 3E1 (CR)
READY          (press the pause button)
S7, 280, 2C0 (CR)
READY          (press the pause button)
S8, 300, 341 (CR)
READY          (press the recorder)
```

Right, now the start address of the FINAL program can be entered followed by the key E. (You can try keys D, N and F later if you like).

The result:

(SP) R

```
DEAR MR/MRS/MISS
JUNIOR!IT'S TIME
TO GO.SEE YOU
AGAIN IN BOOK 4!
```

Appendix 1

The main board plus a single RAM/EPROM card

Simple memory extension

Depending on circumstances, the memory range of the Junior Computer may be extended without adding the interface board. We have already seen how to provide extra memory with the aid of the interface board, but there is an easier method, which is simply to connect a single RAM/EPROM card to the main board. Note that **only one** memory card can be added in this manner, since the expansion connector of the basic Junior Computer is not buffered.

The procedure is as follows:

1. **Link points D and EX on the main board.**
   This enables memory on the RAM/EPROM card to be accessed externally. Select lines K1 . . . K5 are disabled when extra memory is being addressed.
2. **Link either pin 3 or pin 4 of IC8 (the output of gate N5, see figure 25 in chapter 10) on the RAM/EPROM card to the EX connection** (pin 30c on the expansion connector of the main computer board) via the connector on the RAM/EPROM card.

Why is this necessary? The output level of N5 is determined by the input levels of gates N1 and N2. These are the connections V, W, X and Y, one or several of which are used to provide connections to the corresponding pins of the main address decoder, IC5.

Provided the inputs to N1 and N2 are all logic one, their outputs will be logic one and so the output of N5 will be logic zero. This means that point EX is also logic zero, enabling the address decoder, IC6, on the main board to operate as of old: the memory etc. on the main board is accessed. As soon as one of the inputs of N1 or N2 becomes logic zero, because a 4k address block is being accessed on the RAM/EPROM card, the output of N5 will become logic one and the D input of IC6 on the main board will also go high. Now memory situated outside the main board can be accessed. For further details about the operation of the address decoder on the main board see figure 4 and table 1 in chapter 10.

The above leads to a number of important points to remember:

3. **Do not address memory with the pins 'Ø', '1' and 'F' on the RAM/EPROM card if the NMI, RES and IRQ vectors are located in the original EPROM on the main board.** Using these would cause the EX signal to become logic one whenever addresses in the range Øxxx, 1xxx and Fxxx are accessed. This would prevent these locations from being selected correctly on the main Junior Computer board (via IC6).

There is another reason for not connecting pins 'Ø' and '1'. Supposing the operator wishes to fill the gap in the memory map between locations Ø4ØØ and 17FF on the main board with memory stored on the RAM/EPROM card (locations between 18ØØ . . . 19FF can not be used, as this is part of the address range selected by K6). There will always be a 4k block that is partly filled with RAM/EPROM card memory and partly with main board memory. Reading main board memory inside such a 4k block will cause the data buffers on the RAM/EPROM card to be enabled for a 'read' process as well. In other words, there is a danger of multiple addressing, as was described in chapter 10 (see figure 6).

4. The only point to be remembered when fetching the NMI, RES and IRQ vectors from the EPROM located on the main board, is to ensure that this EPROM can be addressed: pin 'F' of IC5 on the RAM/EPROM card **must not** be used in this instance. Address lines A13 . . . A15 will now be inoperative and the vectors are fetched from the original EPROM (page 1F instead of page FF). **Addresses FØØØ . . . FFFF can not be used on the RAM/EPROM card.** This does not matter, as there is still plenty of room left for one card on pages 2Ø . . . EF. This amounts to over 52k of memory, more than the 24k maximum which can be located on a single RAM/EPROM card (8k RAM plus 16k EPROM).
5. In order to fetch the various vectors from EPROM stored on the RAM/EPROM card **pin 'F' must be connected. This means that EPROM will have to be located on page FF.** The address locations involved for the various vectors are:

FFFA for NMIL;
FFFB for NMIH;
FFFC for RESL;
FFFD for RESH;
FFFE for IRQL;
FFFF for IRQH;

This subject is described in full detail in the latter part of chapter 10, where several pages are devoted to the RAM/EPROM card.

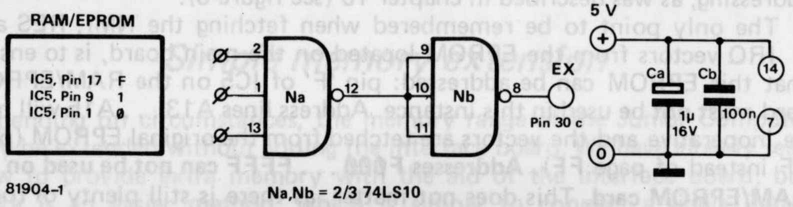
At this stage it may be useful to recap on the **octuple addressing** problem. As you will remember, page Øx of the memory situated on the main board coincides with pages 2x, 4x, 6x, 8x, Ax, Cx and Ex and page 1x coincides with pages 3x, 5x, 7x, 9x, Bx, Dx and Fx (where x = Ø . . . F). Well, whenever an address which does not belong to a 4k block decoded on the RAM/EPROM card is being accessed, the original address decoder on the main board (IC6) is enabled. It all depends on how many of the 4k blocks (or rather how many pins of IC5) on the RAM/EPROM card are being employed. At least, octuple addressing is out of the question here. As far as page 1x is concerned, double addressing or more will certainly occur if the vectors are fetched from the original EPROM.

6. The two boards may be linked with short wires, connectors or a combination of connectors and ribbon cable. Figures 19 and 20 in chapter 10 show how this can be carried out.
7. As only one RAM/EPROM card is involved, it can be powered from the original (unmodified) Junior Computer power supply. However, it may be necessary to install a slightly larger heatsink. Note that if 2708 EPROMs are used, they require +12 V and −5 V supplies in addition to +5 V. On

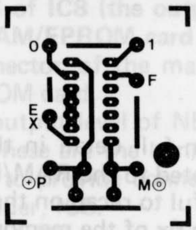
the other hand, 2716 devices only require +5 V. Therefore, 2716 EPROMs should be no problem for the existing power supply.

Method number two

One way to prepare the RAM/EPROM card is to connect the output of N5 on the RAM/EPROM card to pin D of IC6 on the main board by way of connector point 30c (point EX). There is, however, another method which involves the circuit shown in figure 1. This has the advantage that



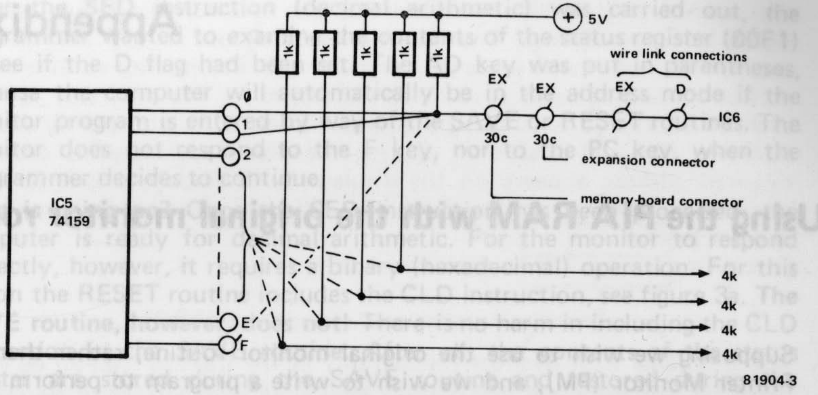
it completely excludes octuple addressing (but not double addressing, which is necessary anyway, when the vectors are located in the original EPROM, so that page 1x coincides with page Fx). It also has a disadvantage: putting the idea into practice is a little more difficult. Either a printed circuit board or a piece of Veroboard will have to be made and mounted. Both are shown in figure 2.



If the vectors are located in the original EPROM, the link to pin 'F' of IC5 may be omitted. Again, avoid connecting pins Ø and 1 to points V, W, X or Y, as the 4k address blocks concerned are not available for full use in any case (part of them are included on the main board). What is more, the data buffers are inclined to read data of their own accord, leading to the multiple addressing headache. Apart from these two aspects, the procedure is the same as before.

Method number three

In addition to the hardware modifications already mentioned, there is yet another method. This is drawn in figure 3. Use a 74159 instead of the 74154 for the main address decoder, IC5, on the RAM/EPROM card. This operates in exactly the same manner as the 74154, except that the outputs, Ø . . . F, are now open collector outputs. This means that each one has to be connected to the +5 V supply via pull-up resistors, so as to provide the

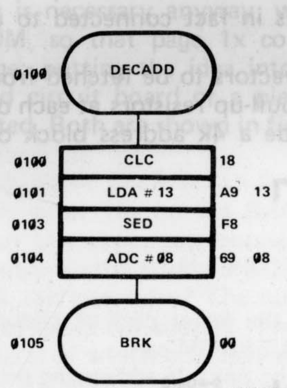


correct logic levels. The main difference being that the outputs of the 74159 can be interconnected whereas the outputs of the 74154 can not. Linking outputs Ø and 1 of IC5 and then connecting them to pin D of IC6 on the main board (by way of the EX line), means that only addresses ØØØØ . . . 1FFF may now be accessed on the main board. By the way, only a certain part of this address range is in fact connected to usable memory or I/O.
Pin F can also be linked up to enable the vectors to be fetched from the original EPROM. Do not forget to mount pull-up resistors at each output (2 . . . E or 2 . . . F) that is used to decode a 4k address block on the RAM/EPROM card.

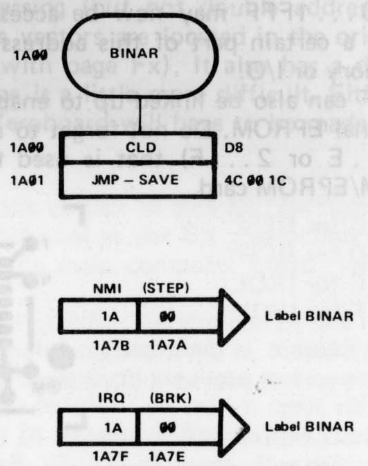
Appendix 2

Using the PIA RAM with the original monitor routine

Supposing we wish to use the original monitor routine, rather than the Printer Monitor (PM), and we wish to write a program to perform a bit of decimal arithmetic. This will involve the use of the instruction SED (F8) to allow the instruction sequence to be executed correctly. An example of this is given in figure 1a in the form of the DECADD routine.



81905-1a



81905-1b

The relevant data is entered, the NMI and IRQ jump vectors are made to point at address 1CØØ (start of the SAVE routine in the monitor program), the STEP switch is turned on and the following keys are depressed:

key	display	
AD Ø 1 Ø Ø GO	Ø1Ø1A9	(instruction CLC is executed)
PC	Ø1Ø1A9	
GO	Ø1Ø3F8	(instruction LDA is executed)
PC	Ø1Ø2F8	
GO	Ø1Ø469	(instruction SED is executed)
(AD) Ø	1Ø4Øxx	
Ø	Ø4ØØxx	
F	Ø4ØØxx	(??)
1	4ØØ1xx	

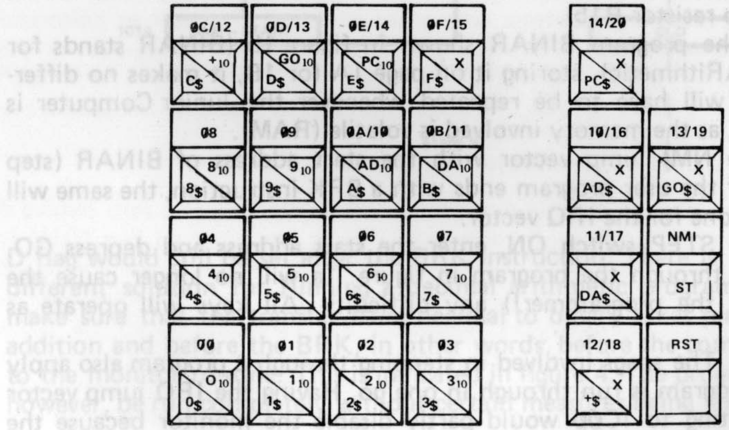
After the SED instruction (decimal arithmetic) was carried out, the programmer wanted to examine the contents of the status register (ØØF1) to see if the D flag had been set. The AD key was put in parentheses, because the computer will automatically be in the address mode if the monitor program is entered by way of the SAVE or RESET routines. The monitor does not respond to the F key, nor to the PC key, when the programmer decides to continue.

What is going on? Once the SED instruction has been processed, the computer is ready for decimal arithmetic. For the monitor to respond correctly, however, it requires a binary (hexadecimal) operation. For this reason the RESET routine includes the CLD instruction, see figure 3a. **The SAVE routine, however, does not!** There is no harm in including the CLD instruction in the SAVE routine. After all, the contents of the status register are stored during the SAVE routine and restored during the GOEXEC routine after the GO key is depressed (see chapter 7 in book 2). So it does not matter if any of the flags are altered during the monitor routine as they are all resorted afterwards.

The consequences

It is not that the monitor program becomes completely useless during a decimal arithmetic operation, contrary to what you might expect from the above. It can be seen from figure 2 that a number of keys have been assigned a different key function. These are keys A . . . F which have now become AD, DA, +, GO and PC. Keys RST and ST will remain the same as before, but the numeric key functions A . . . F are no longer available. The index dollar sign in figure 2 indicates the normal function of the key, whereas 10 indicates a decimal arithmetic operation.

The reason for this distinction is that the monitor works on the basis of the key value of the depressed key. This is illustrated in figure 16 on page 130 of Book 2. The values are first obtained in binary and are then operated on in decimal, as a result of which several are unmodified as shown in figure 2 (the right-hand diagonal of each square).



81905-2

The upshot of it is that the command key functions have now been assigned to different keys. The numeric key functions A . . . F are not available, which means that while the computer is in the monitor routine no addresses containing bytes A . . F can be keyed in. The only way to reach such an address is to enter the address closest to it with purely decimal numbers (Ø . . 9) and then depress the plus key (=key C!) until the required address appears on the display . . . Obviously, this is far from perfect and having command keys in different places only complicates matters further.
Something will have to be done about this!

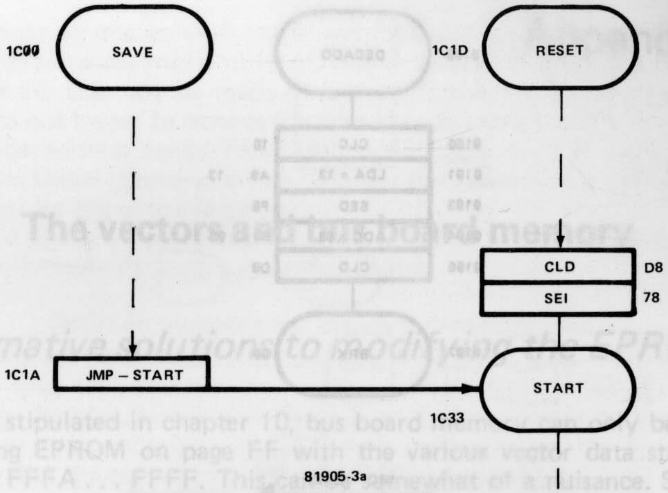
Solution number one: change the EPROM

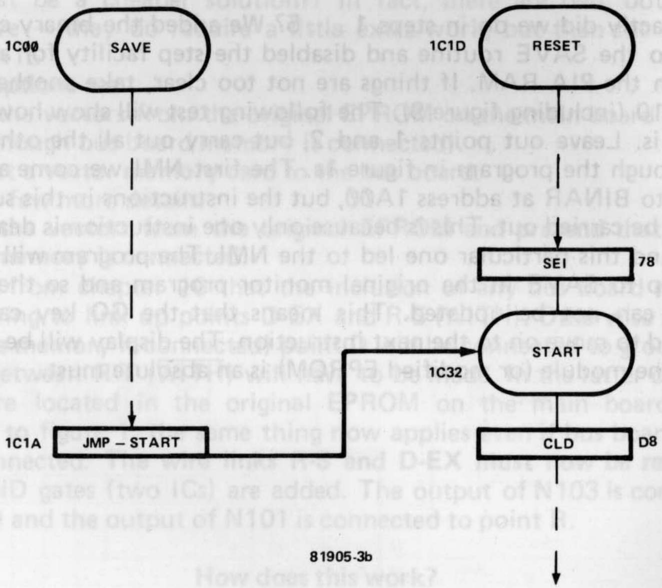
The problems with decimal calculations in the monitor can all be eliminated by shifting a few instructions around as shown in figure 3b. Now, a jump to the SAVE routine automatically leads to a CLD instruction. This can be accomplished by relocating the START label and exchanging the SEI and CLD instructions. The latter instruction will no longer belong to the RESET routine but the main monitor routine, beginning at the START label. The following three memory locations inside the original EPROM will have to be altered:
IC1B: 32 (previously 33)
IC31: 78 (previously D8)
IC32: D8 (previously 78)
The EPROM will therefore have to be reprogrammed. The D version of the EPROM will still be available in its original form, however, as there is another way around the problem.

Solution number two: add a little monitor routine in the PIA RAM

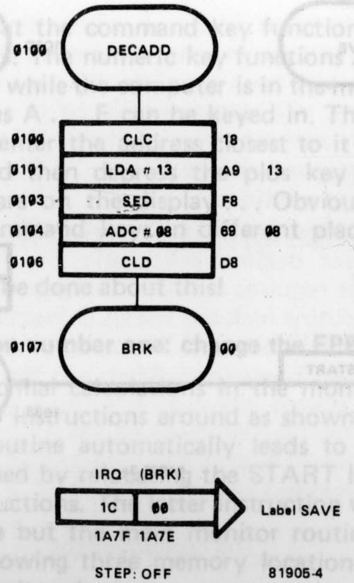
1. Mount the module mentioned in chapter 10 (figures 8b, 11 and 12) on the main board of the Junior Computer. Practical details are given in the relevant text.
2. Link select line K7 of the address decoder, IC6, on the main board to one of the select inputs of the module and link the select line K6 of the address decoder to the other module select input (this link can be connected to resistor R15).
3. Key in the program BINAR shown in figure 1b (BINAR stands for BINary ARithmetic), storing it on page 1A (or 1B, it makes no difference). This will have to be repeated whenever the Junior Computer is switched on, as the memory involved is volatile (RAM).
4. Load the NMI jump vector with the start address of BINAR (step mode). If the user program ends with a BRK instruction, the same will have to be done for the IRQ vector.
5. Turn the STEP switch ON, enter the start address and depress GO. Stepping through the program in figure 1a will no longer cause the monitor (or the programmer!) any difficulty. All keys will operate as normal.

By the way: The snags involved in stepping through a program also apply when the program is run through in one go. Having the IRQ jump vector (BRK) pointing to ICØØ would partly disable the monitor because the





D flag would still be set after the BRK instruction. There is, of course, a different solution for running a decimal arithmetic program. This is to make sure that the transfer from decimal to binary takes place after the addition and before the BRK, in other words, before the computer jumps to the monitor routine. This is indicated in figure 4. The program can not, however, be run completely without certain measures being taken.



What exactly did we do in steps 1 . . . 5? We added the binary correction (CLD) to the SAVE routine and disabled the step facility for a program stored in the PIA RAM. If things are not too clear, take another look at chapter 10 (including figure 9). The following test will show how vital the module is. Leave out points 1 and 2, but carry out all the others. Then step through the program in figure 1a. The first NMI we come across will take us to BINAR at address 1AØØ, but the instructions in this subroutine will not be carried out. This is because only one instruction is dealt with at a time and this particular one led to the NMI. The program will therefore not jump to SAVE in the original monitor program and so the program counter can not be updated. This means that the GO key can not be depressed to move on to the next instruction. The display will be unlit. Hence, the module (or modified EPROM) is an absolute must.

Appendix 3

The vectors and bus board memory

Alternative solutions to modifying the EPROM

As it was stipulated in chapter 10, bus board memory can only be added by locating EPROM on page FF with the various vector data stored at addresses FFFA . . . FFFF. This can be somewhat of a nuisance. Supposing, for instance, that the user is only interested in adding extra RAM in the form of one or more 16k dynamic RAM cards (to be described in book 4). Just for the sake of six EPROM bytes the operator would be obliged to purchase a relatively expensive RAM/EPROM card. Surely, there must be a cheaper solution?! In fact, there are two, both being inexpensive — they do require a little extra work, but then the result is well worth it.

The two options are:

1. Fetch the vectors from the original EPROM on the main board after all (even though bus board memory is connected).
2. Connect a **vector memory card** to the bus board.

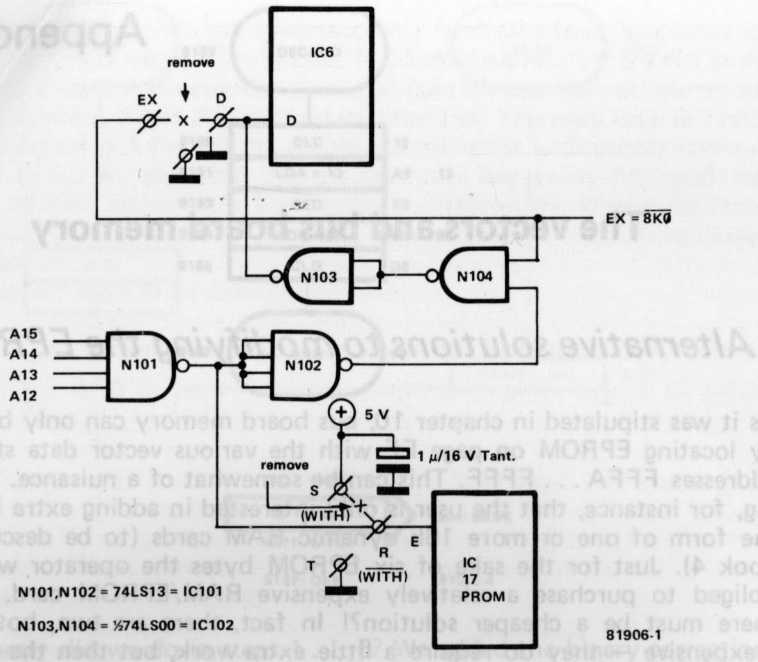
Now for a few more details.

1. Fetch the vectors from the original EPROM and pretend that no bus board memory is connected.

We know from chapter 10 that the inclusion of any bus board memory means having to link up points D-EX and R-S (WITH). Otherwise (that is, if no extra memory is connected) point D will be connected to ground and the link between R-T ($\overline{\text{WITH}}$) will have to be made. In the latter case, the vectors are located in the original EPROM on the main board. Well, according to figure 1, the same thing now applies even if bus board memory is connected. **The wire links R-S and D-EX must now be removed!** Four NAND gates (two ICs) are added. The output of N103 is connected to point D and the output of N101 is connected to point R.

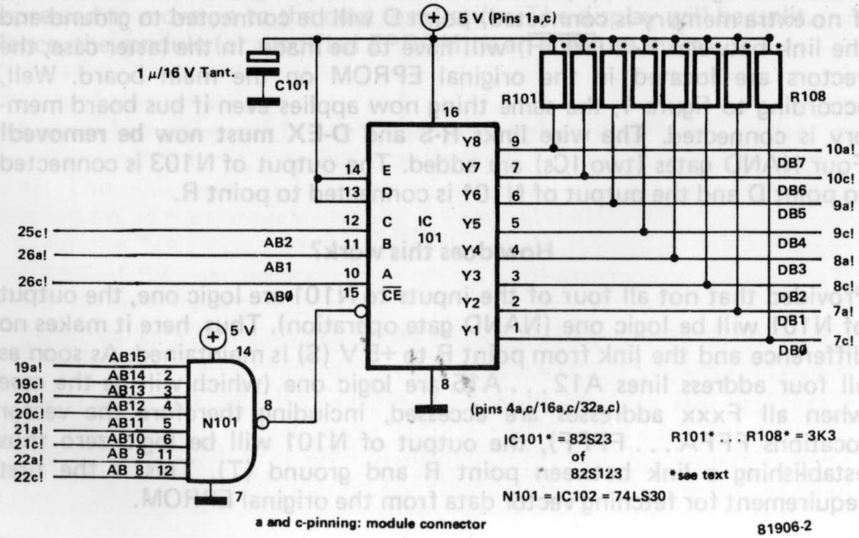
How does this work?

Provided that not all four of the inputs to N101 are logic one, the output of N101 will be logic one (NAND gate operation). Thus, here it makes no difference and the link from point R to +5 V (S) is maintained. As soon as all four address lines A12 . . . A15 are logic one (which will be the case when all Fxxx addresses are accessed, including therefore the vector locations FFFA . . . FFFF), the output of N101 will be logic zero thus establishing a link between point R and ground (T). This is the first requirement for fetching vector data from the original EPROM.



Now for the second requirement. If the four address lines A12 . . . A15 are all logic one, pin D of IC6 on the main board will have to be logic zero. This is achieved by way of the inverter N102 (N101 and N102 combined constitute an AND gate) and the AND gate comprised of N103 and N104. If the 8KØ signal then becomes logic zero when a vector is fetched, point D of IC6 will go low instead of high.

Putting figure 1 into practice will require a certain amount of constructional skill. It is best to build the circuit on a small piece of Veroboard and place it close to the vertical link board between the main and interface boards. The pin assignments of IC101 and IC102 can be found in figure 18 in chapter 10. Use may be made of the non-buffered address lines A12 . . . A15. Do not forget to remove the wire links R-S and D-EX.



Clearly, the address range Fxxx (4k block) may not be used for connecting bus board memory, as the data bus buffers on the interface board are disabled for the entire range.

If you feel that this particular solution is rather tricky, an easier one — albeit slightly more expensive — is given below.

Solution number two: the vector memory card

Take a piece of Veroboard 10 cm wide by 5 or 6 cm long. Mount a male connector (like that shown in figure 19a in chapter 10) on one of the long sides. Place the circuit shown in figure 2 on the board and plug the vector memory card thus obtained into a female connector on the bus board. **Make sure that the connector pins are correctly positioned.**

How does it work?

A 32 byte PROM of the same type as that for the interface board was chosen (if IC101 is an 82S123 type, no pull-up resistors are required). Now, however, all eight bits are employed. The required vector data is programmed into six of the 32 available memory locations. Provided that the chip enable (CE) pin is logic one, the data output of the PROM will be disabled and will not, therefore, affect the data bus. This happens when not all of the address lines connected to the inputs of N101 are logic one. The situation changes radically when the eight address lines are all logic one, which is the case when any address belonging to page FF is being accessed. This would be one of the vector locations FFFA . . . FFFF. Data is now transferred from the PROM to the data bus, provided a read operation is involved. (Whenever vector data is being fetched by the processor, the computer is undergoing a read operation).

The A . . . E address inputs of the PROM are connected to address lines ABØ . . . AB4 respectively. This provides more than enough decoding for the vector memory locations. The PROM must be programmed as follows:

address EDCBA (hex)	data (hex)
ØØ . . . 19	ØØ
1A	2F (NMIL)
1B	1F (NMIH)
1C	1D (RESL)
1D	1C (RESH)
1E	32 (IRQL)
1F	1F (IRQH)

Although the original EPROM is not being used to obtain vectors, it does perform the task of looking after the RESET routine and transferring data from the vector PROM to jump vector stored in PIA RAM (IRQ, BRK and NMI). Since the PROM is being decoded by thirteen address lines, the loss in bus board memory will only concern a single page of 256 (= FF) bytes. Page FF, however, is part of a 4k block which is therefore no longer available for locating RAM/EPROM card memory.

Appendix 4

The hex dump for the TAPE MONITOR (TM) program

```
       Ø  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
Ø8ØØ: 2Ø 72 ØC A5 FA 8D 7Ø 1A A5 FB 8D 71 1A 4C 4E Ø8
Ø81Ø: A2 ØØ 8E 79 1A 8E 7Ø 1A 8E 71 1A 8E 72 1A 8E 73
Ø82Ø: 1A 8E 68 1A A9 Ø6 8D 82 1A A9 1E 8D 83 1A 2Ø 36
Ø83Ø: Ø9 DØ FB 2Ø 36 Ø9 FØ FB 2Ø 36 Ø9 FØ F6 2Ø F9 1D
Ø84Ø: C9 14 DØ ØE 2Ø Ø2 ØB A2 FF EC 79 1A FØ B2 E8 4C
Ø85Ø: 21 Ø8 C9 1Ø DØ Ø8 2Ø DF Ø9 A2 ØØ 4C 21 Ø8 C9 12
Ø86Ø: DØ 11 EE 68 1A AØ Ø9 CC 68 1A DØ C2 AØ ØØ 8C 68
Ø87Ø: 1A FØ BB C9 13 DØ 25 A5 E2 8D 7Ø 1A A5 E3 8D 71
Ø88Ø: 1A A5 E8 8D 72 1A A5 E9 8D 73 1A 2Ø DF Ø9 2Ø D3
Ø89Ø: 1E A9 1E 8D 83 1A A2 FF 9A 4C CA 1C C9 11 DØ Ø3
Ø8AØ: 4C B5 1C Ø6 F9 Ø6 F9 Ø6 F9 Ø6 F9 Ø5 F9 85 F9 AØ
Ø8BØ: ØØ CC 68 1A DØ Ø6 8D 79 1A 4C 2E Ø8 C8 CC 68 1A
Ø8CØ: DØ Ø6 8D 71 1A 4C 2E Ø8 C8 CC 68 1A DØ Ø6 8D 7Ø
Ø8DØ: 1A 4C 2E Ø8 C8 CC 68 1A DØ Ø6 8D 73 1A 4C 2E Ø8
Ø8EØ: C8 CC 68 1A DØ Ø6 8D 72 1A 4C 2E Ø8 C8 CC 68 1A
Ø8FØ: DØ Ø5 85 E3 4C 2E Ø8 C8 CC 68 1A DØ Ø5 85 E2 4C
Ø9ØØ: 2E Ø8 C8 CC 68 1A DØ Ø5 85 E5 4C 2E Ø8 C8 CC 68
Ø91Ø: 1A DØ Ø5 85 E4 4C 2E Ø8 4C 1Ø Ø8 B9 BB Ø9 8D 8Ø
Ø92Ø: 1A 8E 82 1A 98 AØ 7F 88 DØ FD A8 E8 E8 C8 EØ 1Ø
Ø93Ø: DØ E9 2Ø A7 1D 6Ø A9 7F 8D 81 1A A2 Ø8 AØ ØØ CC
Ø94Ø: 68 1A DØ Ø9 AD 79 1A 85 F9 2Ø 1B Ø9 6Ø C8 CC 68
Ø95Ø: 1A DØ Ø9 AD 71 1A 85 F9 AØ Ø4 DØ ED C8 CC 68 1A
Ø96Ø: DØ Ø9 AD 7Ø 1A 85 F9 AØ Ø8 DØ DE C8 CC 68 1A DØ
Ø97Ø: Ø9 AD 73 1A 85 F9 AØ ØC DØ CF C8 CC 68 1A DØ Ø9
Ø98Ø: AD 72 1A 85 F9 AØ 1Ø DØ CØ C8 CC 68 1A DØ Ø8 A5
Ø99Ø: E3 85 F9 AØ 14 DØ B2 C8 CC 68 1A DØ Ø8 A5 E2 85
Ø9AØ: F9 AØ 18 DØ A4 C8 CC 68 1A DØ Ø8 A5 E5 85 F9 AØ
Ø9BØ: 1C DØ 96 A5 E4 85 F9 AØ 2Ø DØ 8E 66 21 7F 7F 52
Ø9CØ: Ø8 Ø9 7F 52 Ø8 47 7F Ø6 Ø8 Ø9 7F Ø6 Ø8 47 7F Ø3
Ø9DØ: Ø6 42 Ø9 Ø3 Ø6 42 47 Ø6 2B 21 Ø9 Ø6 2B 21 47 A9
Ø9EØ: 7D 8D 6C 1A A9 C3 8D 6D 1A A9 Ø3 8D 76 1A A9 Ø2
Ø9FØ: 8D 77 1A A9 47 A2 FF 8D 82 1A 8D 78 1A 8E 83 1A
```

```
ØAØØ: A9 ØØ A2 7F 8D 8Ø 1A 8E 81 1A 8D 6E 1A 8D 6F 1A
ØA1Ø: AD 7Ø 1A 85 FA AD 71 1A 85 FB A2 FF 8E 74 1A A9
ØA2Ø: 16 2Ø A3 ØA CE 74 1A DØ F6 A9 2A 2Ø A3 ØA AD 79
ØA3Ø: 1A 2Ø 8B ØA AD 7Ø 1A 2Ø 7A ØA AD 71 1A 2Ø 7A ØA
ØA4Ø: A5 FB CD 73 1A DØ 23 A5 FA CD 72 1A DØ 1C A9 2F
ØA5Ø: 2Ø A3 ØA AD 6E 1A 2Ø 8B ØA AD 6F 1A 2Ø 8B ØA A9
ØA6Ø: Ø4 2Ø A3 ØA A9 Ø4 2Ø A3 ØA 6Ø AØ ØØ B1 FA 2Ø 7A
ØA7Ø: ØA E6 FA DØ CB E6 FB 4C 4Ø ØA A8 18 6D 6E 1A 8D
ØA8Ø: 6E 1A AD 6F 1A 69 ØØ 8D 6F 1A 98 A8 4A 4A 4A 4A
ØA9Ø: 2Ø 9A ØA 98 29 ØF 2Ø 9A ØA 6Ø C9 ØA 18 3Ø Ø2 69
ØAAØ: Ø7 69 3Ø A2 Ø8 8E 75 1A 4A 48 9Ø ØC 2Ø C8 ØA 2Ø
ØABØ: E5 ØA 2Ø E5 ØA 4C C1 ØA 2Ø C8 ØA 2Ø C8 ØA 2Ø E5
ØACØ: ØA 68 CE 75 1A DØ E1 6Ø AE 76 1A 2C D5 1A 1Ø FB
ØADØ: AD 6C 1A 8D F4 1A AD 78 1A 49 8Ø 8D 82 1A 8D 78
ØAEØ: 1A CA DØ E7 6Ø AE 77 1A 2C D5 1A 1Ø FB AD 6D 1A
ØAFØ: 8D F4 1A AD 78 1A 49 8Ø 8D 82 1A 8D 78 1A CA DØ
ØBØØ: E7 6Ø A9 32 8D 82 1A 8D 78 1A A9 7E 8D 83 1A A9
ØB1Ø: 7F 8D 81 1A A9 ØØ 8D 6E 1A 8D 6F 1A A9 FF 8D 6B
ØB2Ø: 1A 2Ø C2 ØB 6E 6B 1A AD 6B 1A 2Ø E8 ØB C9 16 DØ
ØB3Ø: FØ AØ ØA 8C 69 1A 2Ø 36 ØC 2Ø 5D ØC C9 16 DØ DC
ØB4Ø: CE 69 1A DØ F1 2Ø 36 ØC 2Ø 5D ØC C9 2A FØ Ø6 C9
ØB5Ø: 16 FØ F2 DØ AD 2Ø 5D ØC 2Ø F3 ØB CD 79 1A DØ 3E
ØB6Ø: 2Ø F3 ØB 2Ø 4B ØC 85 FA 2Ø F3 ØB 2Ø 4B ØC 85 FB
ØB7Ø: 2Ø F3 ØB 3Ø 8D FØ 13 2Ø 4B ØC AØ ØØ 91 FA E6 FA
ØB8Ø: DØ Ø2 E6 FB 2Ø 64 ØC 4C 7Ø ØB 2Ø F3 ØB CD 6E 1A
ØB9Ø: DØ Ø9 2Ø F3 ØB CD 6F 1A DØ Ø1 6Ø 4C Ø2 ØB AD 79
ØBAØ: 1A C9 ØØ FØ BB C9 FF DØ F2 2Ø F3 ØB 2Ø 4B ØC 2Ø
ØBBØ: F3 ØB 2Ø 4B ØC AD 7Ø 1A 85 FA AD 71 1A 85 FB 4C
ØBCØ: 7Ø ØB 2C 82 1A 1Ø FB AD D4 1A AØ FF 8C F6 1A AØ
ØBDØ: 14 88 DØ FD 2C 82 1A 3Ø FB 38 ED D4 1A AØ FF 8C
ØBEØ: F6 1A AØ Ø7 88 DØ FD 6Ø 48 A9 36 8D 8Ø 1A 68 2Ø
ØBFØ: 64 ØC 6Ø 2Ø 36 ØC C9 2F DØ Ø1 6Ø 2Ø 19 ØC 3Ø FA
ØCØØ: ØA ØA ØA ØA 8D 6A 1A 2Ø 36 ØC C9 2F FØ EC 2Ø 19
ØC1Ø: ØC 3Ø E7 ØD 6A 1A AØ Ø1 6Ø C9 3Ø 3Ø ØC C9 3A 3Ø
ØC2Ø: ØB C9 14 3Ø Ø4 C9 47 3Ø Ø3 AØ FF 6Ø C9 4Ø 3Ø Ø3
ØC3Ø: 18 69 Ø9 29 ØF 6Ø A2 Ø8 2Ø C2 ØB 6E 6B 1A CA DØ
ØC4Ø: F7 2E 6B 1A 4E 6B 1A AD 6B 1A 6Ø 48 18 6D 6E 1A
ØC5Ø: 8D 6E 1A AD 6F 1A 69 ØØ 8D 6F 1A 68 6Ø 48 49 7F
ØC6Ø: 8D 8Ø 1A 68 48 AD 78 1A 49 Ø2 8D 82 1A 8D 78 1A
ØC7Ø: 68 6Ø 38 A5 FA E9 Ø1 85 FA A5 FB E9 ØØ 85 FB 6Ø
```

Appendix 5

The hex dump for the PRINTER MONITOR (PM) program

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
1000: D8 78 A9 67 8D 82 1A A9 00 8D 80 1A A2 FE 8E 5A
1010: 1A E8 8E 5B 1A 9A 86 F2 A9 7F 8D 81 1A 8D 83 1A
1020: A2 02 8E 59 1A A9 5F 8D 7C 1A A9 10 8D 7D 1A A9
1030: CF 8D 7A 1A A9 14 8D 7B 1A 2C 80 1A 30 FB 20 E0
1040: 12 4E 5F 1A 6E 5E 1A AD 5E 1A 8D 5C 1A AD 5F 1A
1050: 8D 5D 1A A2 08 20 03 13 20 BE 12 C9 7F D0 A1 20
1060: 46 12 20 E8 11 A2 FF 9A 86 F2 20 59 12 20 68 12
1070: 20 AE 12 C9 2B D0 09 20 13 12 20 F8 11 4C 6A 10
1080: C9 2D D0 09 20 1A 12 20 F8 11 4C 6A 10 C9 20 D0
1090: 0E A5 F8 85 FA A5 F9 85 FB 20 F8 11 4C 6A 10 C9
10A0: 2E D0 0F A5 F8 A0 00 91 FA 20 13 12 20 F8 11 4C
10B0: 6A 10 C9 52 D0 13 A6 F2 9A A5 FB 48 A5 FA 48 A5
10C0: F1 48 A6 F5 A4 F4 A5 F3 40 C9 4C D0 52 A0 14 20
10D0: D6 11 A5 F3 20 8F 12 A0 1A 20 D6 11 A5 F4 20 8F
10E0: 12 A0 20 20 D6 11 A5 F5 20 8F 12 A0 26 20 D6 11
10F0: A5 F0 20 8F 12 A5 EF 20 8F 12 A0 2C 20 D6 11 A9
1100: 01 20 8F 12 A5 F2 20 8F 12 A0 32 20 D6 11 20 28
1110: 12 A0 38 20 D6 11 20 F3 11 4C 6A 10 4C B0 11 C9
1120: 50 D0 0E A5 EF 85 FA A5 F0 85 FB 20 F8 11 4C 6A
1130: 10 C9 4D D0 E7 A0 52 20 D6 11 20 87 13 10 03 4C
1140: 5F 10 38 AD 65 1A ED 63 1A AD 66 1A ED 64 1A 90
1150: EE 20 E8 11 A2 06 20 F3 11 CA D0 FA A0 00 98 20
1160: 9B 12 20 F3 11 20 F3 11 C8 C0 10 D0 F1 AD 63 1A
1170: 85 FA AD 64 1A 85 FB 20 E8 11 A2 10 A5 FB 20 8F
1180: 12 A5 FA 20 8F 12 A0 17 20 D9 11 38 AD 65 1A E5
1190: FA AD 66 1A E5 FB B0 06 20 E8 11 4C 5F 10 A0 00
11A0: B1 FA 20 8F 12 20 F3 11 20 13 12 CA D0 DD F0 C7
11B0: C9 47 D0 0B 20 8A 14 30 03 4C 6A 10 4C 5F 10 C9
11C0: 53 D0 08 20 3B 14 30 F4 4C 6A 10 20 6F 12 D0 03
11D0: 4C 70 10 4C 6A 10 20 E8 11 B9 BD 13 C9 03 F0 07
11E0: 20 34 13 C8 4C D9 11 60 A9 0D 20 34 13 A9 0A 20
11F0: 34 13 60 A9 20 4C EF 11 20 E8 11 A5 FB 20 8F 12
1200: A5 FA 20 8F 12 20 F3 11 A0 00 B1 FA 20 8F 12 20
1210: F3 11 60 E6 FA D0 02 E6 FB 60 38 A5 FA E9 01 85
1220: FA A5 FB E9 00 85 FB 60 A5 F1 8D 67 1A A2 08 0E
1230: 67 1A 90 09 A9 01 20 9B 12 CA D0 F3 60 A9 00 20
```

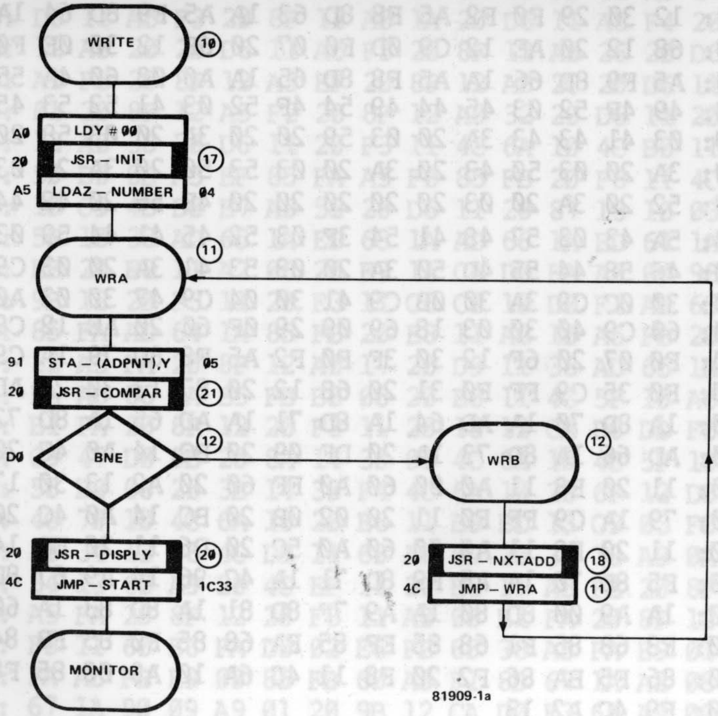
```
1240: 9B 12 CA D0 EA 60 A0 00 20 D6 11 20 E8 11 60 A0
1250: 07 4C 48 12 A0 0E 4C 48 12 A0 00 8C 63 1A 8C 64
1260: 1A 8C 65 1A 8C 66 1A 60 A0 00 84 F8 84 F9 60 20
1270: 1E 14 30 10 A2 04 06 F8 26 F9 CA D0 F9 05 F8 85
1280: F8 A0 00 60 A0 46 20 D6 11 20 E8 11 A0 FF 60 48
1290: 4A 4A 4A 4A 20 A4 12 20 34 13 68 29 0F 20 A4 12
12A0: 20 34 13 60 C9 0A 18 30 02 69 07 69 30 60 2C 80
12B0: 1A 30 FB 8E 61 1A A2 08 20 03 13 20 12 13 2C 80
12C0: 1A 10 0A 38 6E 62 1A CA D0 F1 4C D4 12 18 6E 62
12D0: 1A CA D0 E7 20 12 13 AD 62 1A 29 7F AE 61 1A 60
12E0: 18 AD 5A 1A 69 01 8D 5A 1A AD 5B 1A 69 00 8D 5B
12F0: 1A 2C 80 1A 10 EA AD 5A 1A 8D 5E 1A AD 5B 1A 8D
1300: 5F 1A 60 AD 5C 1A 8D 5E 1A AD 5D 1A 8D 5F 1A 4C
1310: 1E 13 AD 5A 1A 8D 5E 1A AD 5B 1A 8D 5F 1A 38 AD
1320: 5E 1A E9 01 8D 5E 1A AD 5F 1A E9 00 8D 5F 1A EA
1330: EA B0 EB 60 8E 60 1A 8D 62 1A AD 82 1A 29 FE 8D
1340: 82 1A 20 12 13 A2 07 4E 62 1A 90 30 AD 82 1A 09
1350: 01 8D 82 1A 20 12 13 CA D0 ED AE 59 1A AD 82 1A
1360: 09 01 8D 82 1A 20 12 13 CA D0 F2 2C 80 1A 10 04
1370: AE 60 1A 60 2C 80 1A 10 FB 6C 7C 1A AD 82 1A 29
1380: FE 8D 82 1A 4C 54 13 20 AE 12 C9 2C F0 07 20 6F
1390: 12 30 29 F0 F2 A5 F8 8D 63 1A A5 F9 8D 64 1A 20
13A0: 68 12 20 AE 12 C9 0D F0 07 20 6F 12 30 0E F0 F2
13B0: A5 F9 8D 66 1A A5 F8 8D 65 1A A0 00 60 4A 55 4E
13C0: 49 4F 52 03 45 44 49 54 4F 52 03 41 53 53 45 4D
13D0: 03 41 43 43 3A 20 03 59 20 20 3A 20 03 58 20 20
13E0: 3A 20 03 50 43 20 3A 20 03 53 50 20 3A 20 03 50
13F0: 52 20 3A 20 03 20 20 20 20 20 4E 56 20 42 44 49
1400: 5A 43 03 57 48 41 54 3F 03 52 45 41 44 59 03 48
1410: 45 58 44 55 4D 50 3A 20 03 53 41 3A 20 03 C9 30
1420: 30 0C C9 3A 30 0B C9 41 30 04 C9 47 30 03 A0 FF
1430: 60 C9 40 30 03 18 69 09 29 0F 60 20 AE 12 C9 2C
1440: F0 07 20 6F 12 30 3F F0 F2 A5 F8 8D 79 1A C9 00
1450: F0 35 C9 FF F0 31 20 68 12 20 87 13 30 28 AD 63
1460: 1A 8D 70 1A AD 64 1A 8D 71 1A AD 65 1A 8D 72 1A
1470: AD 66 1A 8D 73 1A 20 DF 09 20 BC 14 A0 4C 20 D6
1480: 11 20 E8 11 A0 00 60 A0 FF 60 20 A2 13 30 17 8D
1490: 79 1A C9 FF F0 11 20 02 0B 20 BC 14 A0 4C 20 D6
14A0: 11 20 E8 11 A0 00 60 A0 5C 20 D6 11 20 EB 14 30
14B0: F5 8D 70 1A A5 F9 8D 71 1A 4C 96 14 A9 67 8D 82
14C0: 1A A9 00 8D 80 1A A9 7F 8D 81 1A 8D 83 1A 60 85
14D0: F3 68 85 F1 68 85 EF 85 FA 68 85 F0 85 FB 84 F4
14E0: 86 F5 BA 86 F2 20 F8 11 4C 6A 10 A9 00 85 F8 85
14F0: F9 4C A2 13
```

Appendix 6

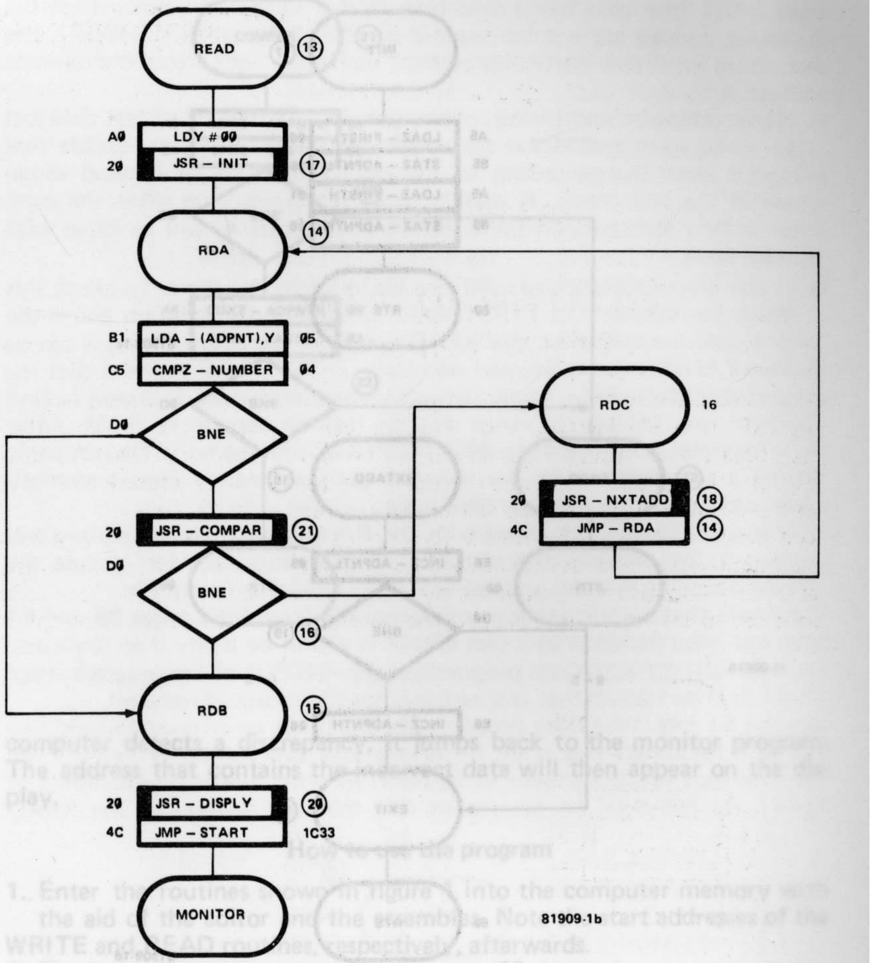
RAM test program

Test the reading and writing skills of your Junior Computer

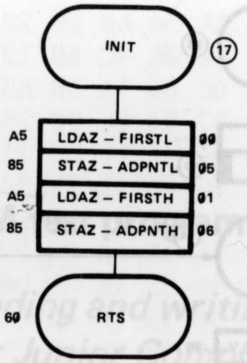
Any expansion of the Junior Computer must necessarily involve an extension of the available RAM capacity. This can be done either on the interface board or by using a bus board with one or more RAM/EPROM cards and/or 16k dynamic RAM cards connected to it. It is absolutely vital that all the RAM locations can be written to and read from correctly.
The program shown in figure 1 tests whether the read and write operations are being carried out successfully. The program should be entered into the original RAM on the main board by means of the editor and the assembler routines in the monitor program. The programmer has a certain amount of freedom in his/her choice of start address, which explains the labels provided in figure 1. For the test program to be stored in original RAM, it follows that the basic section of the Junior Computer should be working perfectly.
The program itself is split up into several routines: a write routine (figure 1a), a read routine (figure 1b) and four subroutines (figures 1c . . . 1f). During the write operation, all the memory locations from FIRST to LAST are loaded with data which is specified in memory location NUMBER. Once data has been loaded into all the memory locations in the test block, the LAST address will appear on the display, including the data contained in location NUMBER. The READ routine in figure 1b enables all the memory locations in the test block to be read (provided, of course, the program is entered correctly!). The data which is read is then compared with the written data (the contents of NUMBER). As soon as the



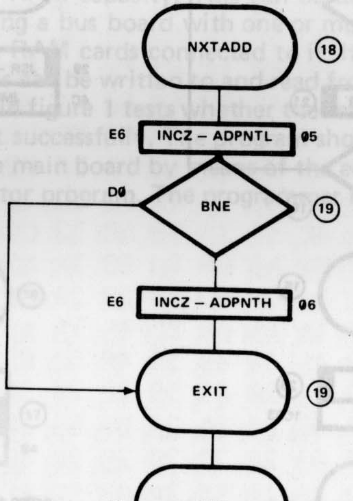
81909-1a



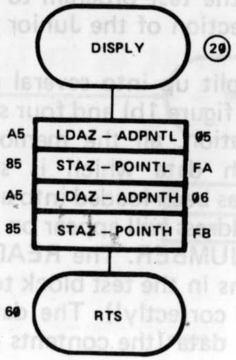
81909-1b



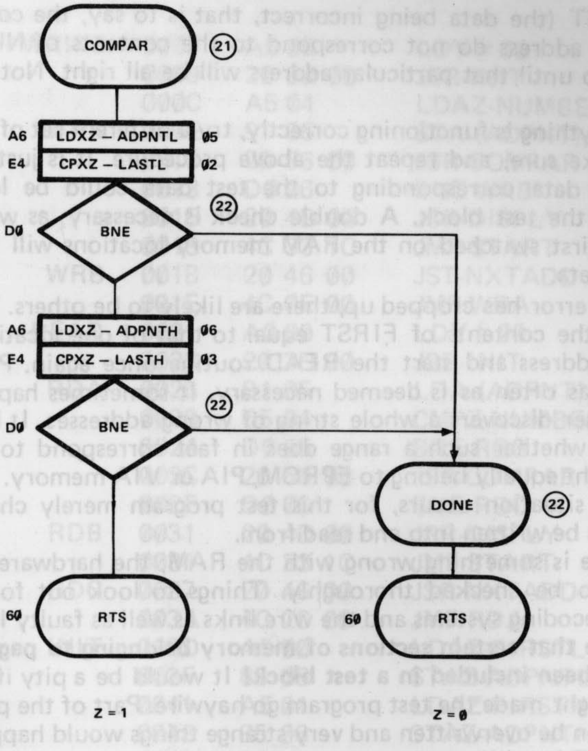
81909-1c



81909-1d



81909-1e



81909-1f

computer detects a discrepancy, it jumps back to the monitor program. The address that contains the incorrect data will then appear on the display.

How to use the program

1. Enter the routines shown in figure 1 into the computer memory with the aid of the editor and the assembler. Note the start addresses of the WRITE and READ routines, respectively, afterwards.
2. The following memory locations on page ØØ are used:

ØØØØ	FIRSTL	first address in test block
ØØØ1	FIRSTH	first address in test block
ØØØ2	LASTL	last address in test block
ØØØ3	LASTH	last address in test block
ØØØ4	NUMBER	test data
ØØØ5	ADPNTL	read/write address pointer
ØØØ6	ADPNTH	read/write address pointer

Enter the FIRST and LAST addresses according to the choice of test block. It is preferable not to use ØØ or FF is a actual test data.
3. Start the WRITE routine. At the end of this the last address plus the test data is shown on the display.
4. Start the READ routine. If at the end of this the last address of the test block is indicated on the display, all the RAM belonging to that test block is all right. If, on the other hand, an address appears which is lower

than LAST (the data being incorrect, that is to say, the contents of the displayed address do not correspond to the contents of NUMBER), the section up until that particular address will be all right. Note the relevant address.

5. If everything is functioning correctly, try different set of test data just to make sure, and repeat the above procedure. It is just feasible that incorrect data corresponding to the test data could be located somewhere in the test block. A double check is necessary, as when the computer is first switched on the RAM memory locations will be filled with random data.

6. If one error has cropped up, there are likely to be others. To check this make the contents of FIRST equal to that of one location above the 'wrong' address and start the READ routine once again. Point 6 can be repeated as often as is deemed necessary. It sometimes happens that the programmer discovers a whole string of wrong addresses. It is well to find out first whether such a range does in fact correspond to RAM. After all, it might equally belong to EPROM, PIA or VIA memory. Do not panic if such a situation occurs, for this test program merely checks memory which can be written into and read from.

7. If there is something wrong with the RAM, the hardware involved will have to be checked thoroughly. Things to look out for include the address decoding systems and the wire links as well as faulty ICs.

Make sure that certain sections of memory belonging to pages ØØ and Ø1 have not been included in a test block! It would be a pity if an unnecessary oversight made the test program go haywire. Part of the program range would then be overwritten and very strange things would happen!

Use the ST key. When the programmer hits upon an error and would like to check for others (see point 6), he/she may depress the ST key and call the interrupt routine shown in figure 2. In other words, the contents of FIRST do not have to be altered and the start address of the READ

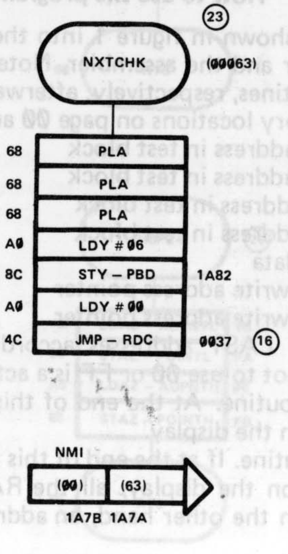


Table 1.

```
WRITE   ØØØ7  AØ ØØ      LDY#ØØ
        ØØØ9  2Ø 3D ØØ   JSR-INIT
        ØØØC  A5 Ø4      LDAZ-NUMBER
  WRA   ØØØE  91 Ø5      STA-(ADPNT),Y
        ØØ1Ø  2Ø 56 ØØ   JSR-COMPAR
        ØØ13  DØ Ø6      BNE WRB
        ØØ15  2Ø 4D ØØ   JSR-DISPLY
        ØØ18  4C 33 1C   JMP-START
  WRB   ØØ1B  2Ø 46 ØØ   JST-NXTADD
        ØØ1E  4C ØE ØØ   JMP-WRA
 READ   ØØ21  AØ ØØ      LDY#ØØ
        ØØ23  2Ø 3D ØØ   JSR-INIT
  RDA   ØØ26  B1 Ø5      LDA-(ADPNT),Y
        ØØ28  C5 Ø4      CMPZ-NUMBER
        ØØ2A  DØ Ø5      BNE RDB
        ØØ2C  2Ø 56 ØØ   JSR-COMPAR
        ØØ2F  DØ Ø6      BNE RDC
  RDB   ØØ31  2Ø 4D ØØ   JSR-DISPLY
        ØØ34  4C 33 1C   JMP-START
  RDC   ØØ37  2Ø 46 ØØ   JSR-NXTADD
        ØØ3A  4C 26 ØØ   JMP-RDA
 INIT   ØØ3D  A5 ØØ      LDAZ-FIRSTL
        ØØ3F  85 Ø5      STAZ-ADPNTL
        ØØ41  A5 Ø1      LDAZ-FIRSTH
        ØØ43  85 Ø6      STAZ-ADPNTH
        ØØ45  6Ø         RTS
NXTADD  ØØ46  E6 Ø5      INCZ-ADPNTL
        ØØ48  DØ Ø2      BNE EXIT
        ØØ4A  E6 Ø6      INCZ-ADPNTH
 EXIT   ØØ4C  6Ø         RTS
DISPLY  ØØ4D  A5 Ø5      LDAZ-ADPNTL
        ØØ4F  85 FA      STAZ-POINTL
        ØØ51  A5 Ø6      LDAZ-ADPNTH
        ØØ53  85 FB      STAZ-POINTH
        ØØ55  6Ø         RTS
COMPAR  ØØ56  A6 Ø5      LDXZ-ADPNTL
        ØØ58  E4 Ø2      CPXZ-LASTL
        ØØ5A  DØ Ø6      BNE DONE
        ØØ5C  A6 Ø6      LDXZ-ADPNTH
        ØØ5E  E4 Ø3      CPXZ-LASTH
        ØØ6Ø  DØ ØØ      BNE DONE
 DONE   ØØ62  6Ø         RTS
NXTCHK  ØØ63  68         PLA
        ØØ64  68         PLA
        ØØ65  68         PLA
        ØØ66  AØ Ø6      LDY#Ø6
        ØØ68  8C 82 1A   STY-PBD
        ØØ6B  AØ ØØ      LDY#ØØ
        ØØ6D  4C 37 ØØ   JMP-RDC
```

routine does not have to be re-entered. After the ST key is depressed, the READ routine will continue from label RDC. This section prepares the computer for reading the next address location. The program will stop whenever it encounters incorrect data at a certain address. This should then be noted and the ST key depressed once more. This method does have one slight disadvantage: the programmer may inadvertently pass the final address in the test block, LAST, by depressing the ST key too often. As soon as LAST appears in the display, stop depressing the ST key!

A different approach

Instead of using the method outlined above, the RAM can also be tested with the aid of the monitor routine. Whenever a numeric key is depressed in the data mode, the instruction STA - (POINTL), Y is carried out. During the multiplexing of the display the instruction LDA - (POINTL), Y is carried out at regular intervals. This is none other than a write operation followed by a read operation.
The next step:
AD x x x x   xxxx yy
DA z z   xxxx zz
Here, xxxx stands for an address belonging to the test block. The data zz must be chosen to be completely different from yy. If after typing in the data no zz appears, something must have gone wrong. The whole test block may be examined in this manner: depress the plus key and note any erroneous data, etc.
P.S. An assembled version of the test program shown in figures 1 and 2 (including the ST key routine, NXTCHK) is provided in table 1. The program was located on page ØØ. The start address of WRITE is ØØØ7, that of READ is ØØ21 and the ST key routine starts at address ØØ63. Do not forget to make sure the NMI jump vector is pointing to this latter address! Also, make sure that the FIRST address is higher than ØØ6F (the end of the program)!

Appendix 7

ASCII character code.

character	ASCII-code (hexadecimal)	character	ASCII-code (hexadecimal)
Ø	3Ø	V	56
1	31	W	57
2	32	X	58
3	33	Y	59
4	34	Z	5A
5	35	space	2Ø
6	36	•	2E
7	37	(	28
8	38	+	2B
9	39	¦	7C
A	41	&	26
B	42	!	21
C	43	$	24
D	44	*	2A
E	45	)	29
F	46	;	3B
G	47	— or ~	7E
H	48	—	2D
I	49	/	2F
J	4A	,	2C
K	4B	%	25
L	4C	– or ←	5F
M	4D	>	3E
N	4E	?	3F
O	4F	:	3A
P	5Ø	#	23
Q	51	@	4Ø
R	52	'	27
S	53	=	3D
T	54	"	22
U	55	<	3C

[illegible] does not have to be re-entered. After the ST key is depressed, the [illegible] continue from label RDC. This section prepares the computer for reading the next address location. The program will stop whenever it encounters incorrect data at a certain address. This should then be noted and the ST key depressed once more. This method does have one slight disadvantage: the programmer may inadvertently pass the final address in the test block, LAST, by depressing the ST key too often. As soon as LAST appears in the display, stop depressing the ST key!

A different approach

Instead of using the method outlined above, the RAM can also be tested with the aid of the monitor routine. Whenever a numeric key is depressed in the data mode, the instruction STA - (POINTL), Y is carried out. During the multiplexing of the display the instruction LDA - (POINTL), Y is carried out at regular intervals. This is none other than a write operation followed by a read operation.
The next step:
AD x x x x xxxx yy
DA z z xxxx zz
Here xxxx stands for an address belonging to the test block. The data zz must be chosen to be completely different from yy. If after typing in the data yy or zz appears, something must have gone wrong. The whole test block may be examined in this manner: depress the + key and note any erroneous data, etc.
P.S. An assembled version of the test program shown in figures 1 and 2 (including the ST key routine, NXTCHK) is presented in table 1. The program was located on page 00. The start address of WRITE is 0007, that of READ is 0021 and the ST key routine starts at address 0063. Do not forget to make sure the NMI jump vector is pointing to this latter address! Also, make sure that the FIRST address is higher than 000F (the end of the program)!

Book 3 describes a number of steps that need to be taken to transform the single-board, basic Junior Computer into a complete personal computer system. This involves adding an interface board to allow the machine to communicate with the outside world (its operator) in an 'adult' manner. The interface board provides additional I/O, a cassette interface, an RS 232 interface and an internal connection with the buffered bus board.

The hardware extensions complete the physical development of the computer and the sophisticated software in Book 4 enables the machine to fully utilise its additional memory capacity by extending its communication skills. As a result, a number of peripheral devices, such as a printer or a video terminal, can be 'hooked up' to the computer.

Elektor Publishers Ltd.
Canterbury

ISBN 0905705 09 2