# A scientific computer — 3

## Construction, testing and operating

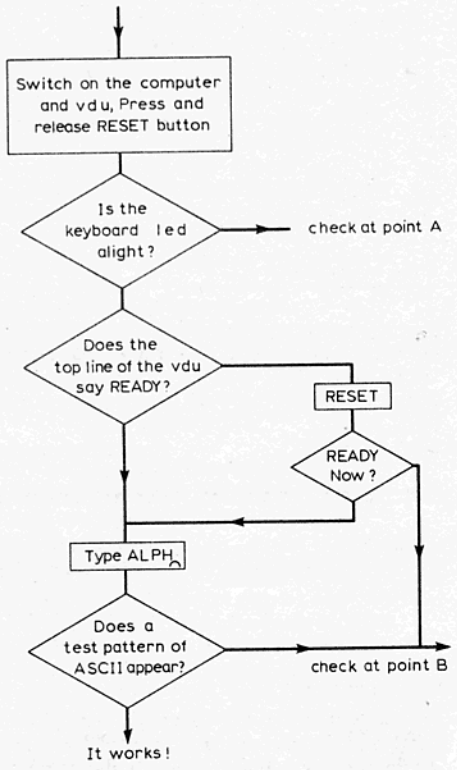
by J. H. Adams, M.Sc.

**ALTHOUGH** this is not a simple project, with careful soldering and the usual m.o.s. precautions, the construction should be quite straightforward. It is worthwhile building the power supplies first and testing them under load conditions of 3A for the +5V and 0.5A each for the −5V and +12V, until the regulators have reached their working temperatures. As a power supply failure can be particularly damaging, a generous heatsink, especially on the 2N3055, is recommended.

The next section to build should be the v.d.u. circuit, which will provide the video and sync signals required in the development of the display interface. To ease later work, the interface should be built as described in part 2. With the character generator and the 21L02s left out, and with the variable resistor set to a maximum, a correct display will consist of 32 rows of 64 oblongs. With the character generator and memories in place, these oblongs will become rows of random ASCII characters. When this is displayed, the variable resistor is reduced to move the display up the screen until it is as high as possible with correct linearity of all 32 lines. Reducing the resistor too much will either cramp or expand the top line and eventually wrap it back into what will then become visible fly-back. Table 2 gives test points and their waveforms for the v.d.u., and table 3 gives processor checks.

Once the circuit has been completed it should be thoroughly checked. A particularly devastating fault occurs if power lines appear on the wrong i.c. pins, especially the outputs of t.t.l. circuits. An ohm-meter, connected between each of the supplies in turn and the i.c. pins, will check for this kind of fault. Cautious constructors need only insert IC18 and IC26 out of the memory devices, the first r.o.m. and the r/w.m. covering 1C00 to 1FFF, for the initial test. Fig. 17 gives a suitable sequence for these tests.

The computer requires an ASCII coded input, comprising 7 bits of inverted data, together with a positive strobe pulse, active during the presence of the code at the computer input buffer. The Carter type 756 keyboard will give such signals when connected as shown in Fig. 18. For those constructing a purpose-built keyboard, DEL, ESC, CTRL and — are not required, and RS should carry the legend ↗S the legend ↓. The l.e.d. lights whenever the Z80 is in the halt state and indicates that the computer is waiting for keyboard data.



Switch on the computer and vdu, Press and release RESET button
Is the keyboard led alight?
check at point A
Does the top line of the vdu say READY?
RESET
READY Now?
Type ALPH⌢
Does a test pattern of ASCII appear?
check at point B
It works!

**Fig. 17.** *Test sequence for the computer.*

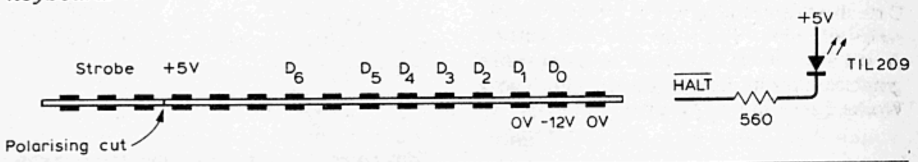


**Fig. 18.** *Connection for the Carter 756 keyboard.*

**Table 2 v.d.u. test points and waveforms**

| Location | Waveform | Possible remedy |
|---|---|---|
| $IC_{28c}$ out | 8MHz clock, t.t.l. | $IC_{28}$ or crystal |
| $IC_{32a}$ out | 4μs pulses every 64μs | check back through $IC_{48}$, $IC_{30}$, $IC_{29}$ |
| $IC_{35}$ pin 11 | approx 50Hz, t.t.l. | check through $IC_{35}$, $IC_{29}$, $IC_{34}$, and then, |
| $IC_{36}$ pins 1, 6 | approx 100μs pulses every 20ms | check $IC_{36}$; pin 6 is normally low |
| $IC_{37}$ pin 2 | 16μs pulses every 64μs | check $IC_{31b}$, $IC_{48}$ |
| $IC_{45}$ pin 1 | 1.333MHz with 8MHz bursts | check $IC_{37}$, $IC_{33}$ |
| $IC_{45}$ pin 7 | 48μs bursts of data every 64μs | check $IC_{45}$ |
| Video | mixed video and blanking information | check $IC_{33}$ |
| Sync. | mixed syncs | check $IC_{32b}$ and the differentiating network |

## Using the computer

When the assembled computer has been tested, the r.o.ms and at least IC blocks 22 and 26 should be inserted. Two programs, one in the low-level and one in the high-level language, are programmed into some spare space at the end of the third r.o.m., and these will be used to demonstrate the computer's ways.

In the tables and explanations of commands and program lines, the symbol ⌢ means that a space has to be typed at that point, e.g.

TAPE ⌢ 1800 1880

means that you type TAPE space 1800 1880. As explained earlier, this is one of the bases of the systems operation.

## Low level operation

When you have a high-level language, working in machine code may seem like talking in Morse code. However, low-level programs, if properly written, usually occupy less memory space, run faster and allow the computer to be used as a controller of processes, as well as a calculating machine. Table 4 lists the computer's machine code commands. At the address 0B16, there is an example of a code-breaking game where the computer makes up a four digit number using the digits 1 to 8, and then marks your attempts to guess the code by awarding black symbols for correct digits in the correct place and white symbols for any remaining digits in the code which are in the wrong place. With the computer in the READY state, type

RUN ⌢ 0B16

and then your first guess at the code, say 1234. The computer will mark your guess and wait for your next attempt. Note that, as soon as you type something, in this case, the first letter of RUN, the READY disappears, and does not return until you break the code, indicating that the program has finished running. The READY state may be achieved at any time by pressing RESET, or by typing FS. To examine the code set during the program, return to the READY state and type

LIST ⌢ 1FE4

The computer will then list from address 1FE4 to 21CF. The format used for listing gives the address of the first byte, which will appear on that line, and then the remainder up to the end of the row of 16, spaced in blocks of four for easy inspection. When a line is broken into, as in this case, the computer maintains the layout by indenting the top line by the correct amount. The first four bytes contain the computer's code, a 00 representing a digit 8.

The game may be played over and over again, using the command

RUN ⌢ 0B16

but, as an illustration, suppose that the program is to be simplified. To alter the program, it must first be copied into the r/w.m. so type

MOV ⌢ 0B16 0C00 1E16

which will move the program out into r/w.m. and, because some of the bytes in the program relate to the memory area that the program occupies, type

COR ⌢ 1E16 1F00 0B 1E

The computer will reply

1E24 1E37 1E3D 1E5B 1E62

meaning that it found 0Bs at these addresses, and changed them to 1Es. Now list the program,

LIST ⌢ 1E16

and note that the byte at 1EAC, i.e. the 13th byte on the row starting 1EA0, is a 77. Type

ALT ⌢ 1EAC 33

at which the 77 will change to a 33. This will limit the number range in the code from 1 to 4, rather than 1 to 8 as in the original. The computer will not return to the READY state because often more than one modification is carried out at a time and, as in this case, the 0B at 1E5B, which was altered in the COR command, was not part of an address to be altered, but is the k in the word black, and so must be changed back with

1E5B 0B

Now, press FS to achieve the READY state and

RUN ⌢ 1E16

to play the simpler form of game.

Using the machine code is essentially a matter of practice and experience, but more details will appear in part 4.

## High level language

Table 5 lists the BURP statements, and

**Table 3. Processor checks.**

| Location | Waveform | Possible remedy |
|---|---|---|
| Point A IC$_1$ pin 18 | low | If it is low, test the l.e.d. |
| Point B IC$_1$ pin 6 | 8MHz | If not, check clock buffer circuit around IC$_{14b}$. |
| Address bus | Various | $A_0$ to $A_6$ should be cycling through refresh addresses, $A_7$ to $A_{15}$ should be low, except for $A_8$, which carries a 500kHz square wave. A line not conforming to this pattern is not necessarily at fault. Check for levels between 0.8 and 2.4V, as these imply a short to one of the other address lines. |
| $\overline{RD}$ | 750ns pulses every 2μs | These are active low, and so the pulses described are to a low state. Repeat the test for shorts. |
| $\overline{WR}$ | High | |
| $\overline{MREQ}$ | 750 then 500ns pulses every 2μs | |
| $\overline{IORQ}$ | High | |
| Data bus | Various | During the $\overline{RD}$ pulse, the computer accesses the memory, although, as it is the HALT state, (when working correctly) the Z80 ignores the accessed byte. The accessed address is 0357, which puts the byte E6$_H$ or 11100110 on lines $D_7$ to $D_0$ respectively during the pulse. If these lines are tested, short circuits must not be confused with tri-state periods, when the lines may float into the intermediate voltage range. |

**Table 4. Machine code commands.**

| | |
|---|---|
| ALPH ⌢ | Produces an alphanumeric test pattern on the v.d.u. |
| *ALT ⌢ XXXX YY | Changes the contents of location XXXX to YY |
| COR ⌢ XXXX YYYY AA BB | Scans XXXX to YYYY-1 inclusive and alters any AA to BB. |
| FILL ⌢ XXXX | See note 3. |
| FIND ⌢ XX YY | Finds the consecutive bytes XX YY and lists the addresses at which they occur. |
| LIST ⌢ XXXX | Lists the contents of the memory from address XXXX up to a full v.d.u. screen. |
| LOAD ⌢ XXXX | Loads hexadecimal data at XXXX, using the same display format as in list. To load ASCII directly, type a [ and to return type a ] after which, the computer gives the next byte's address and continues to load hexadecimal data. To leave LOAD, type @ which gives a full listing of what has just been loaded, or press RESET or the FS key to regain command. |
| MOV ⌢ XXXX YYYY ZZZZ | Moves the block XXXX to YYYY-1 inclusive to the area of memory beginning with the address ZZZZ. |
| PRINT ⌢ XXXX | Lists from XXXX on the second output device |
| PROM ⌢ | Used in conjunction with the e.p.r.o.m. programmer, this programs the block of data at 1C00 to 1CFF inclusive into the sector of the 2708 selected on the programmer. |
| READ ⌢ XXXX | Reads from tape into memory, starting at location XXXX. READ must be terminated by pressing any key, once the tape has been read in. |
| RUN ⌢ XXXX | Runs from address XXXX. |
| TAPE ⌢ XXXX YYYY | Records, on tape, a short leader of stop bits, followed by the data at locations XXXX to YYYY-1 inclusive and a short trailer of stop bits. |

*Having accepted the alteration, the computer lists out from the previous LIST command starting address. Normally, a LIST of the area in which the alteration is to take place will have been carried out immediately prior to using ALT, and so the altered byte will change to its correct contents on the v.d.u. screen as well as in the memory. After an alteration, the computer does not return to the command state, but waits for any further alterations, typed in as

XXXX YY

and so on. To return to the command state, type FS or press RESET.

**Notes**

1. Take care when using MOV. MOV 1D00 1E00 1CFF will work, and move the block 1D00 to 1DFF inclusive, forward one byte in memory, but MOV ⌢ 1D00 1E00 1D01 will copy 1D00 into 1D01, then 1D01 into 1D02 etc., leaving you with a block of identical characters and your original data lost. While this can sometimes be useful for filling out a block with a particular byte, to do this properly requires a MOV of the block to a separate, vacant, area and then a MOV to 1D01.

2. The PROM ⌢ command takes about 40s to program the e.p.r.o.m. sector completely, and during this time the computer is fully occupied.

3. If less than 256 bytes are to be programmed into an e.p.r.o.m. sector, and the other must be left blank for later additions to the e.p.r.o.ms contents, or, if you wish to add this later bit to an already partly filled e.p.r.o.m., FFs must be present at the bytes which are not to be programmed. This can be achieved by using the extra command FIL ⌢ XXXX, which will fill from XXXX to the next YY00 with

FFs, either before loading into the p.r.o.m. area, or, after loading, to mask off the other unused bytes up to 1CFF. This command also makes programs easier to study on the v.d.u. as it can be used to mask off the rubbish following a program.

4. When loading ASCII, do not try to include a ] in the string of characters as it will terminate your ASCII mode of loading. Also, do not type in any further [ as these will be used in a future adaption to graphics, whose firmware is already in the 2708. Ordinary parentheses, (and) are quite acceptable to the computer.

**Table 5. Burp statements.**

| | |
|---|---|
| INPUT ⌢ A ⌢ B ⌢ etc. | Inputs and assigns one or more variables. |
| LET ⌢ X=A ⌢ SIN ⌢ SQ ⌢ etc. | Assigns the value computed in the expression following the = sign to X. |
| IF ⌢ X=Y ⌢ THEN ⌢ 50 ⌢ | If the condition (which may be <, = or >) is met, then go to line 50. Otherwise, continue. |
| FOR ⌢ X=1 ⌢ STEP ⌢ B ⌢ UNTIL ⌢ C ⌢ | X takes the value 1, the lines up to the line NEXT X are then executed. X is then increased by B and the lines executed again, and this continues until X is greater or equal to C, at which point the computer carries on through the line NEXT X to the next one. |
| NEXT ⌢ X ⌢ | |
| GOSUB ⌢ 200 ⌢ | Goes to line 200 and executes from there until the line RETURN is found, and then returns |
| RETURN ⌢ | to the line following GOSUB. GOSUBs may appear within GOSUB blocks. |
| HALT ⌢ | Halts execution until any key is pressed. |
| TOP ⌢ | Clears, and resets the PRINT position to, the top line of the display area. |
| ERASE ⌢ | As TOP, but it clears the whole display area. |
| END ⌢ | Stops execution and returns to the command state. |
| GO ⌢ 25 ⌢ or GOTO ⌢ 25 ⌢ | Executes from line 25. |
| WRITE ⌢ . . . . | As PRINT for the second output device. |
| PRINT ⌢ . . . . | |
| In prints, the following may appear | |
| An ⌢ | A, printed with n figures after the decimal point and then spaces for the blanked characters, 13 in all. For less than an 8 digit mantissa, the last figure is rounded if necessary. |
| A ⌢ | A, printed with the same number of figures after the decimal point as the previously printed variable or, if it is the first one to be printed, to four figures. This four can be altered in r.o.m. location 0818 by programming 01 to 07 in place of the 04. |
| An, ⌢ | As An ⌢ but with the blanked figures completely suppressed and, if the number is in scientific notation, with the exponent and exponent sign against the mantissa. |
| A, ⌢ | As A ⌢ but with the suppression described above. |

Summarising, without the comma, printed figures always occupy 13 screen locations and thus columns of results will be tabulated no matter what the magnitude of the number. With the comma, alphanumeric data (see below) and variables may be printed in the same line without large gaps appearing.

| | |
|---|---|
| "PRINTED TEXT" ⌢ | Prints the actual characters within the quotes, implying that quotes must not appear in the string of characters. |

It is not possible to have a second (or subsequent) FOR . . . NEXT block within a FOR . . . NEXT block, because the single on-chip memory in the MM57109 is used as a loop counter in conjunction with the NEXT line. These loops, if required, can be set up using for example, in place of the FOR line given,

```
20 ⌢ LET ⌢ X = 1 ⌢           replaces the FOR
21 ⌢                  }
22 ⌢                  }      lines within FOR and NEXT
23 ⌢                  }
24 ⌢ LET X=X ⌢ B ⌢ + ⌢
25 ⌢ IF ⌢ X <C ⌢ THEN ⌢  21 ⌢    replaces the NEXT
```

Table 6 gives the mathematical expressions for LET statements. With the computer in the READY state, type

MOV ⌢ 0BB5 0C00 0C00

and then change to the high level language by pressing RS on the keyboard. The word READY will then be replaced by BURP. The RS key types in RUN ⌢ 0800, and initiates the high level system. The low level MOV command moves the sample program in r.o.m. out into the r/w.m., where it can be examined by typing

LIST ⌢ 5 ⌢

which gives

```
005 FOR A=1 STEP 1 UNTIL 25—
006 LET L=A LOG—
007 PRINT A0 L8 —
008 NEXT A—
009 END—
OC4A
```

The dash shows where a line ends and virtually every term, including the last on each line, is followed by a space. The address 0C4A gives the upper limit of the program storage currently in use, and from 0C00 up to 1DC0 is available. Now type

RUN ⌢ 5 ⌢

The computer should print the common logarithms of the numbers 1 to 25. When it has finished, the computer is ready for a command, indicated whenever BURP is the only word on the top line. Type

DEL ⌢ 6 ⌢

and the program will list out with line number 6 deleted. Note that the end address is now 0C3B, i.e. when lines are deleted, the computer reworks the remaining lines back towards the start of the memory space. This makes best use of the memory and stops the build up of rubbish within the memory which would slow down the program execution. Next,

ADD ⌢
6 ⌢ LET ⌢ L=A ⌢ ROOT ⌢ :

After typing the colon the word ADD will disappear, i.e. you are back in command. The colon is necessary at the end of an ADD or a LOAD because it inserts the hex byte C0 at the end of the program block. This code tells the computer where to stop and go back from, when it is scanning through the memory. Now,

RUN ⌢ 5 ⌢

which will list out the square roots of the numbers 1 to 25. Then,

DEL ⌢ 6 ⌢
ADD ⌢
6 ⌢ LET ⌢ L=A ⌢ EX ⌢ :
RUN ⌢ 5 ⌢

This program lists the natural anti-logs, $e^x$, of the numbers 1 to 25, and will show how the display switches over to scientific notation, the last result being $7.2004907\times10^{10}$. Although mathematically correct, these are rather crude presentations of the results. Type

ADD ⌢
4 ⌢PRINT ⌢"⌢ X ⌢⌢⌢⌢⌢⌢⌢⌢⌢⌢⌢⌢⌢⌢⌢ EXP X":
RUN ⌢ 4 ⌢

which adds a heading above each of the columns, or,

```
          DEL ‿ 7 ‿
            ADD ‿
7 ‿PRINT ‿"THE NATURAL ‿ANTI-
    LOG ‿OF" AO, ‿" ‿IS"L4 ‿:
            RUN ‿ 5 ‿
```

which gives a different display format, see Table 7. Note that the comma after A0 suppressed the characters after the decimal point, rather than leaving a large gap. L4 means L, printed to 4 decimal figures, although without the compaction of the scientific results that a comma would bring. Try

```
          DEL ‿ 7 ‿
            ADD ‿
7 ‿ PRINT ‿ "THE NATURAL. ANTI-
    LOG ‿OF" A0 ‿" ‿IS"L4, ‿:
            RUN ‿ 5 ‿
```

to see the difference. If you make a mistake in these exercises, just terminate the line with a RETURN and type it in again. It is important, as already explained, that the LOAD and ADD commands are only left with a colon, do not be tempted to do so with an RS. If you have corrected a line in this way, when back in the command state, delete that line, and the computer will erase the first line it comes to with that number and then re-list with the second version in the correct place. Naturally, if you have mis-typed a line twice or more, this deleting procedure must be repeated until the correct line appears in place. The running of a program may be halted at any time by pressing any key on the keyboard, but as this returns the computer to the low-level, READY state, follow it with an RS for BURP.

**Table 6. Mathematical expressions for LET statements.**

| Expression | Effect |
|---|---|
| + — * / | Y+X→X, Y—X→X, Y×X→X, Y/X→X. In these four operations, the stack collapses thus; Z→Y, T→Z, 0→T. |
| YX | Y to the power of X→X. Stack collapses as above. |
| REC | 1/X→X, i.e., reciprocal of X. In this, and the following, Y, Z and T remain unchanged. |
| ROOT | $\sqrt{X}$→X |
| SQ | $X^2$→X |
| TENX | $10^x$→X i.e., common anti-logarithm. |
| EX | $e^x$→X i.e., natural anti-logarithm. |
| LN | 1n(X)→X i.e., natural logarithm of X. |
| LOG | log(X)→X i.e., common logarithm of X. |
| SIN | sine (X)→X All trig. functions operate in degrees. |
| COS | cosine (X)→X |
| TAN | tangent (X)→X |
| SIN- | $\sin^{-1}$(X)→X |
| COS- | $\cos^{-1}$(X)→X |
| TAN- | $\tan^{-1}$(X)→X |
| DTR | Converts X in degrees to radians |
| RTD | Converts X in radians to degrees |
| NEG | —X→X i.e., change sign. |
| PI | 3.1415927→X |
| ENT | X→Y, Y→Z, Z→T. T is lost, X remains in X. |
| ROLL | Y→X, Z→Y, T→Z, X→T. Nothing is lost. |
| XEY | X exchanges with Y. |

In use, all of these expressions are followed by a space, e.g. for

$$X=\frac{1}{2\pi\sqrt{LC}}$$

LET ‿ X=L ‿ C ‿ * ‿ ROOT ‿ 2 ‿ * ‿ PI ‿ * ‿ REC ‿

Errors will occur in calculations under certain conditions;
LN or LOG, when X is less than or equal to zero.
TAN when X is an odd multiple of 90° (90°, 270°, 450° etc.)
SIN, COS or TAN when |X| is greater or equal to 9000°
SIN- or COS- when |X| is greater than 1 or less than $10^{-50}$
ROOT when X is negative
/ or REC when X = 0
or for any result less than $10^{-99}$ or greater than $9.9999999 \times 10^{99}$

**Table 7. Print out giving natural anti-logs of numbers 1 to 25. The display switches to scientific notation at number 19.**

```
THE NATURAL ANTI-LOG OF 1. IS 2.7183
THE NATURAL ANTI-LOG OF 2. IS 7.3891
THE NATURAL ANTI-LOG OF 3. IS 20.0855
THE NATURAL ANTI-LOG OF 4. IS 54.5982
THE NATURAL ANTI-LOG OF 5. IS 148.4132
THE NATURAL ANTI-LOG OF 6. IS 403.4288
THE NATURAL ANTI-LOG OF 7. IS 1096.6332
THE NATURAL ANTI-LOG OF 8. IS 2980.9580
THE NATURAL ANTI-LOG OF 9. IS 8103.0839
THE NATURAL ANTI-LOG OF 10.  IS 22026.466
THE NATURAL ANTI-LOG OF 11.  IS 59874.143
THE NATURAL ANTI-LOG OF 12.  IS 162754.79
THE NATURAL ANTI-LOG OF 13.  IS 442413.40
THE NATURAL ANTI-LOG OF 14.  IS 1202604.3
THE NATURAL ANTI-LOG OF 15.  IS 3269017.4
THE NATURAL ANTI-LOG OF 16.  IS 8886110.7
THE NATURAL ANTI-LOG OF 17.  IS 24154953.
THE NATURAL ANTI-LOG OF 18.  IS 65659970.
THE NATURAL ANTI-LOG OF 19.  IS 1.7848       08
THE NATURAL ANTI-LOG OF 20.  IS 4.8517       08
THE NATURAL ANTI-LOG OF 21.  IS 1.3188       09
THE NATURAL ANTI-LOG OF 22.  IS 3.5849       09
THE NATURAL ANTI-LOG OF 23.  IS 9.7448       09
THE NATURAL ANTI-LOG OF 24.  IS 2.6489       10
THE NATURAL ANTI-LOG OF 25.  IS 7.2005       10
```

## Loading programs

Programs are loaded by typing

LOAD ‿

and then the lines of the program, each of which must start with the number of that line. These lines do not have to be entered in the correct order, nor do all three digits of the number need to be typed in as they appear in the list. For internal reasons of the computer, it is not possible to have lines 0, 192, or 237. It is recommended that, for speed of execution, the lines used are kept fairly close together numerically, as this saves the computer scanning for lines which do not exist. In program development, it helps to initially use every third line number so that there is plenty of room for later additions. Remember that LOAD starts loading at the beginning of the program storage area and will thus erase any previously stored programs. If you want to add to the present lines, use ADD.

## Entering data

When the computer comes across the program line INPUT, it goes to the next clear line on the v.d.u. and waits for you to enter the number of variables specified in the program line. Numbers entered must be followed by a space, except in the case of scientifically expressed numbers, which, because of the fixed length of the exponent, are recognised as terminated when the second exponent digit has been typed in. The l.e.d. associated with the keyboard is useful because it indicates whether the computer is, or is not, waiting for you to do something.

Finally, remember the spaces required during loading, and those after the three factors you type in during program execution.

*To be continued*