IN THE BEGINNING ...

Once upon a time, when the world and computers were young, there was no operating system. Indeed, there was no operating assistance at all : when you bought your computer, you might also receive a few *utility programmes* to do standard jobs like printing from card decks, copying files, and so on. Apart from that, you would have an assembler, and perhaps a compiler, and the rest was up to you. And that included the operating. (And if you built your own computer you were really on your own. This was not as unusual as you might imagine; one of the early leaders of the British computer industry was the Lyons catering group, famed for its chain of tea shops. The Lyons Electronic Office built a computer, and it worked well enough that LEO computers were constructed and sold in significant numbers.)

People's expectations of computers were, by modern standards, very limited. A book published in 1959 for commercial programmers[HIS1] was mainly about assembly language programming, though it grudgingly admitted the existence of a thing called Flow-Matic (an ancestor of Cobol), and Fortran.

To illustrate what was involved, here are some excerpts from a programming manual [HIS2] which tell you how to compile and run a Fortran programme on an IBM 1620 computer with paper tape input and output. We've extracted these instructions from a composite set covering both a paper-tape system and a punched-card system, so ignore occasional references to "decks" and other card-related objects. Also ignore the reference to step 7 – we've left that part out because it adds nothing new to the description. (NOTICE that this is a manual for programmers, not for operators.)

Producing the Object Program

The FORTRAN program is available in two forms, card or paper tape. Both forms are divided into two sections; the processor and the subroutines. The sequence of operations that follows is written for both card and paper tape systems.

Eight basic steps are required for producing the object program. These eight steps are summarized below, followed by additional detailed information for steps l, 2, 6, 7, and 8.

1. Clear core storage to zeros.
2. Set the console program switches for compilation.
3. Set the overflow check switch to PROGRAM. and all other check switches to STOP.
4. Press the reset key.
5. For the card system, prepare the card punch for operation by loading blank cards into the punch hopper and by pressing the punch start key.
   For the paper tape system, prepare the paper tape for operation.
6. Load the compiler program deck or tape.
7. Enter the source program statements. These may be read in through the card reader, the paper tape reader, or typed in at the console typewriter.
8. If required, load the subroutine deck or tape.

(Step l)

Clearing Core Storage to Zero

A suggested method for clearing core storage to zeros is:

1. Press the reset key.
2. Press the insert key.
3. Type the instruction 16 00010 00000.
4. Press the release key.
5. Press the start key.
6. After all storage positions have been cleared, press the instant stop key.

(Step 2)

Switch Settings

During compilation of the source program, the console program switches perform the following functions:

	ON	OFF
Switch 1	Causes the source statements to be typed on the console typewriter as they are processed. The first 5-digit field is the object program address of the first instruction compiled for the source statement.	Source statements are not listed.
Switch 2	Causes trace instructions to be compiled.	Trace instructions are not compiled
Switch3	Input to the compiler (source statements) is being entered via the console typewriter.	Source program entered from card reader or paper tape reader.
Switch 4	This switch is used in conjunction with switch 3 when switch 3 is ON. It provides the ability to restart the typing of a statement if you have made an error. Switch 4 is normally OFF. When a typing error is made in a source statement and it is to be corrected, this switch is turned ON, the release and start keys are pressed, and then switch 4 turned OFF. The statement can now be retyped.	

Loading the Compiler

(Step 6)

Paper Tape System

To load the compiler tape, the following procedure must be followed:

1. Mount the compiler tape on the paper tape reader.
2. Press the insert key.
3. Type the instruction 36 00000 00300.
4. Press the release key.
5. Press the start key.

(Step 7)

When the compiler has been successfully loaded, the following instructions are typed on the console typewriter:

ENTER SOURCE PROGRAM, PUSH START

Compilation of the Source Program

To begin compilation after the compiler has been loaded, either press the start key or manually insert the instruction 49 00402.

Two methods of source program input may be used under control of program switch 3, as follows:

1. If input is in cards (switch 3 off), place the source program deck in the read hopper and press the reader start key. If input is in paper tape mount the source program tape on the paper tape reader.
2. If the source program is to be entered from the typewriter (switch 3 on), the compiler will transfer control to the console to await the first statement. After a statement has been typed, press the record mark key and then press the release and start keys to continue compilation. The carriage will return after each statement has been processed, to await the entry of the next statement until an END statement is entered.

After an END statement is processed, the following instruction message is typed on the console typewriter:

SWl ON FOR SYMBOL TABLE, PUSH START

If a typed listing of the symbol table, that was developed during compilation, is not desired, turn off program switch 1. If the listing is required, turn on switch 1.

To continue processing, press the start key.

The following message is typed next, whether the symbol table has been typed or not.

SWl OFF TO IGNORE SUBROUTINES, PUSH START

If the subroutines are to be included in the object program deck or tape, turn on program switch 1, load the subroutine deck or tape, and press the start key. If the subroutine deck or tape is to be read in when the object program is run, switch 1 must be turned off.

To complete the processing, press the start key.

If program switch 1 is off, the following message will be typed:

PROCESSING COMPLETE

(Step 8)

Loading the Subroutines

Under control of program switch 1, as previously described, the subroutine deck or tape may be loaded immediately after compilation, or loaded when the object program is loaded.

Paper Tape System

When operating with the paper tape system, mount the subroutine tape, and load it by pressing the start key. (When restarting, you may insert the instruction 36 00000 00300, press the release key, and then press the start key.) If the source program has used any of the relocatable subroutines, they will either be punched out into the object program if the subroutines are read in immediately after compilation, or they will be loaded into core storage if the subroutines are processed at object time.

After the subroutines have been processed, the following message will be typed on the console typewriter

PROCESSING COMPLETE

Execution of the Object Program

Paper Tape System

When operating with the paper tape system, the object program may be processed immediately after compilation by mounting the object tape and pressing the start key.

The object tape may also be entered by pressing the insert key, typing the instruction 36 00000 00300, and pressing the release and start keys.

If the subroutines are already contained in the object deck or tape, the following message will be typed after the object program has been loaded, and the machine will halt:

LOAD DATA

To initiate the execution of the object program, press the start key on the 1620 console, or manually insert the instruction 49 08300.

SO WHAT'S ALL THAT ABOUT ?

Not many people use their computers like that any more, but it's interesting to look at what was going on throughout that sequence of operations, if only because most of it still happens. The details have changed, but the principles are the same, and in some ways this very low-level description gives a clearer picture of what goes on than does the much more automatic operation of modern machines.

Look first at the description as a whole. It is very clearly written as a sequential programme, with the first section as the main programme, and step 1, step 2, and so on as subroutines. There are frequent conditional instructions, commonly associated with the "console program switches", and at least two examples of iteration – item 6 of step 1, and item 2 of "Compilation of the Source Program". That makes it a reasonably ambitious high-level programming language ! It's written fairly informally, but that's simply because it runs on a comparatively intelligent processor – someone who wants to run a Fortran programme. Nevertheless, the sophistication of the processor doesn't change the essentially mechanical nature of the operation.

The *user interface* is noteworthy. The big difference between the 1620 interface and modern versions is the absence of a screen – mainly on the grounds of expense. To convey information to the computer, one used a typewriter keyboard, the "console program switches" which could be read by programmes, and several special-purpose keys – which is a good deal more than was available on most time-sharing terminals. Output came mainly through the console typewriter, augmented by a display of certain machine registers; item 6 of step 1 worked because the contents of a memory address register were displayed, and you could see it return to address zero when all memory had been cleared. (Incidentally, the typewriter was a real typewriter, operated by solenoids, and you could see the keys bounce, apparently driven by invisible fingers, when the computer was typing messages.)

The paper-tape reader and punch were not used for operating details, but only as channels for comparatively large volumes of information. Another important part of the user interface was therefore *off-line* in the form of equipment for preparing paper tape from a keyboard and for transcribing it onto paper.

Just to make the point, consider the input side of the user interface as it appeared to many people in the early days. It was called a coding form; it was a piece of paper on which was printed a $N \times 80$ grid (where N was often some curious number like 23), on which you

wrote what you wanted to be punched onto *N* punched cards. We emphasise that this isn't a joke. The coding form worked as a user interface for a long time, and you can discuss it in the same way as other, more direct, interfaces. It's still widely used : every time you fill in a form where have to enter items in little boxes, you are probably using a variant of the same interface.

As a final comment, it will perhaps be clear that using computers in this way was *real* interactive computing. *You* went to the computer to run *your* programme, and you knew all about it, and you did everything. It was very rewarding, and enormous fun, and took a lot of time. We shall see that it was the time – or, rather, the comparative idleness of the expensive machinery during that time – which led to the developments we shall be concerned with throughout the course.

HOW IT FITS IN.

The IBM1620 wasn't the first comparatively widespread computer, but it illustrates an important stage in development. It was one of the first (comparatively) cheap computers, which opened up the possibility of computing to organisations with (comparatively) modest budgets. Earlier machines had been more expensive, requiring elaborate facilities and permanent extensive programming and engineering support, or they were frankly experimental. Machines like the IBM1620 initiated the spread of computing which has continued ever since. It is not too far from the mark to suggest that the subsequent history of computing has been dominated by the need to find more profitable ways of using the IBM1620 and its descendants. Certainly, this requirement motivated the development of operating systems, so it will direct the story of the next few chapters.

REFERENCES.

HIS1 : D.D. McCracken, H. Weiss, T.-H. Lee : *Programming Business Computers* (Wiley, 1959).

HIS2 : *Reference Manual, IBM 1620 Fortran* (IBM, 1962).

QUESTIONS.

What's wrong with that mode of working ? Be specific.

What facilities do we need to make it better ?

Would you have recognised those needs at the time ? as a programmer ? as a manager ?
