THE HATFIELD POLYTECHNIC

BBCX

(a hypothetical computer)

Part 1

ADVISORY UNIT FOR
COMPUTER BASED
EDUCATION

I M P O R T A N T

A user enters the BBCX sub-system by typing the PDP-1Ø monitor command:

.BBCX

and leaves by typing the BBCX monitor command:

*LOGOUT

After LOGOUT, the system asks if the user minds his error statistics being recorded. He should type 'NO' to this. At this point he leaves the BBCX sub-system and the PDP-1Ø asks him to CONFIRM: and he must now respond to this (with F, say) or type control-C

THE HATFIELD POLYTECHNIC

Advisory Unit for Computer Based Education

Department of Computer Science

BBCX (a hypothetical computer)

A user's manual written by

W. TAGG

PART 1

Printed and published by The Hatfield Polytechnic

P R E F A C E

This book describes a hypothetical computer which has been simulated on the PDP-1Ø at The Hatfield Polytechnic. As computers are now sophisticated pieces of machinery, they are not suitable as vehicles for introducing Computer Science from a machine code standpoint. Because of this, it has become fashionable to initiate this aspect of the work via a simple but non-existent computer and to provide practical work via a simulation of this machine on a real computer. One advantage which such a solution offers is that the real computer can provide much more in the way of helpful diagnostics.

In the past, such hypothetical computers have fallen into one of two classes. Either they have been based on an ill-defined computer (like the code used for the City and Guilds 319 examination) or they have been so simple that they have not been suitable for problem solving. BBCX (Beginner's Basic Computer - PDP-1Ø) attempts to bridge the gap between these extremes. This has been achieved by choosing a computer which remains suitable for an initial course but which is, perhaps, more advanced in its design than its contemporaries.

BBCX is a fully defined timesharing computer with its own order code, its own instruction execution cycle and with its own "manufacturer's software". Most of this software is written as a suite of programs within the simulated machine. Throughout the course, students are introduced to the monitor, the assembler and the compiler. Although the emphasis is on the results that this software produces for the use rather than how the software itself is written, it is useful for him to be able to study the code (at assembly level) of parts of this software, if only so that he can appreciate that it is essemtially the same as the programs he writes himself.

During 1969 and 197Ø, the author of this manual had the privilege of serving on a working party of the United Kingdom Co-ordinating Committee for Examinations in Computer Science. Part of the duties of this working party was to make recommendations concerning the study and examination of low level programming and machine architecture. The design of "SIMULAC" and its associated software was accepted as a yardstick against which other teaching systems should be measured and there are indications that these recommendations are beginning to be accepted.

All of the ideas incorporated into BBCX have been fully tested in the classroom. During the last five years the system and its predecessors have been used by Computer Science undergraduates, 'A' level Computer Science students and groups of teachers. This manual has been written so that selected chapters can be used directly with 4th or 5th form groups taking a short appreciation course. Other chapters are designed more for the specialist taking one of the new 'A' level examinations in Computer Science. No advanced mathematical knowledge is assumed but it is expected that students using the manual will probably have followed one of the modern 'O' level courses. It is particularly useful for would-be students to be familar with the concepts of binary arithmetic, flowcharting and elementary Boolean algebra.

It is important that I should acknowledge the large amount of support that I have received during the creation of BBCX. Many of the ideas are not my own and much of the work in the programmers office and the classroom has been done by others. For this current version, I am particularly indebted to Pete White who has implemented the simulator for BBCX as well as some of the software. Others who have been particularly involved this time include Bill Freeman (now at York University and my daughter Clare, who started the compiler as an 'A' level project and who finished it as an employee of the Computer Centre. Finally, I should like to thank the Polytechnic for its financial support.

W. TAGG

August 1971

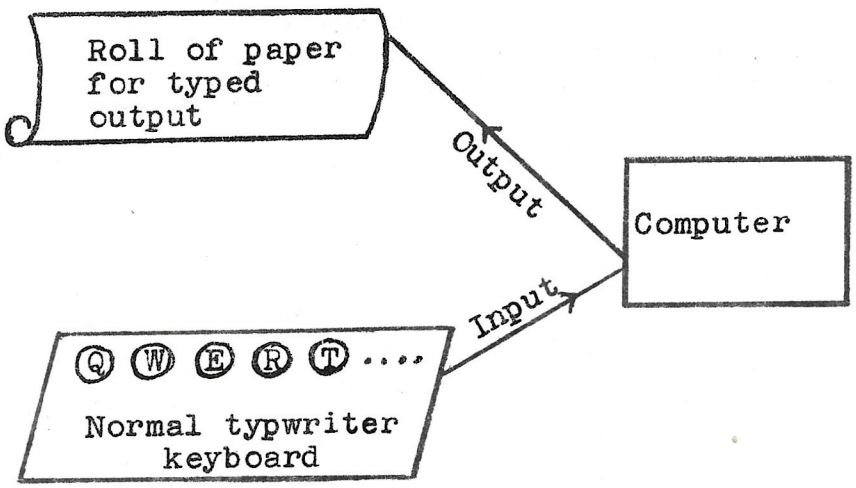
C O N T E N T

Chapter 1 Introduction

The computer described here is a hypothetical or 'paper' computer. Although it does not exist, there is no reason why a computer should not be built to this specification. The PDP - 1Ø computer at the Hatfield Polytechnic has been programmed so that it will behave exactly like the hypothetical machine; this manual has been written for users of this implementation. The design of the machine has had educational requirements in mind and for this reason it differs from a real machine in a number of respects. Most of these differences are unimportant and are no greater than the differences that exist between one real machine and another In a later chapter we shall consider some of those differences in detail but for the moment we shall concentrate on two or three features of the machine and learn to use them properly. One other aspect of the BBCX machine which needs mentioning here is its capability of dealing with more than one user simultaneously. Computers that do this, must not make all of their facilities available to the ordinary user; there has to be some protection so that one user's program is not able to interfere with another user's. When reading the next few chapters, therefore, it should be bourne in mind that there is more to the computer than the part the user has access to. In particular, there are built in routines which are automatically entered when a user attempts to do something which is clearly mistaken.

Input and Output

There must always be some method of passing information from user to machine and from machine to user. In our case, an on-line teleprinter is normally used for this purpose.



This set-up is not particularly convenient as it stands because it implies that the user is only able to see the typed output from the computer and has no record of his own input.

However, it is usual for the user's input to be echoed back but it is important to remember that this echo is by courtesy of the computer only.

When the user types on the teleprinter, the printed copy he obtains is only given for his benefit, what really matters is the electric signals which are sent to the computer. Electric signals pass in the opposite direction when the computer is to output information and these signals cause corresponding messages to be automatically typed for the user to see.

Communicating with the computer

The computer is there to process data. Before it is able to process anything for you, it is necessary for you to formulate your ideas into a program. A program is a 'receipe' which is given to the computer and which controls the way the computer works. Getting this program into the computer is a complicated job but fortunately, when you sit at the terminal the computer has already been programmed to a large extent so that at least it is capable of dealing with the input of your program.

Later, we will examine the 'manufacturer's programs' but to start we will just take them for granted. It is sufficient if you remember that these programs are processing information in exactly the same way as your programs.

The Monitor

This is the most important program in the computer. It has a number of functions, two of the more significant ones are listed here:

i) It keeps the different users seperate and arranges for them all to have a fair ration of time.

ii) It enables the user to communicate in a general way with the computer so that he is able to call on many of the services that are available.

Associated with the monitor program is the command language.

This is a high level language (i.e. it is a long way removed from the natural language of the computer itself) and is used to control the computer in a general way. Instructions given to the computer via the command language cause the computer to take some action immediately. Altogether, there are twenty-six different commands. Each consists of a single word written at the beginning of a line followed by an expression, word or number thus:

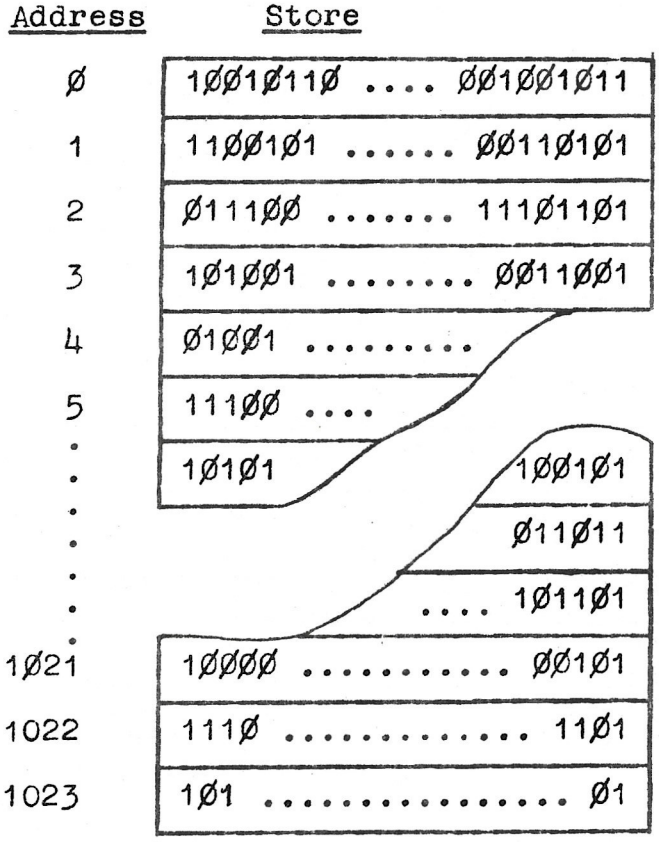
```
LIST    125,275
```

The Assembler

This program is associated with the actual instructions which the computer is built to understand (called machine code). Before explaining the details, it is necessary to learn a little more about the computer itself.

The Store

One of the most important parts of a computer is its store. Basically, this consists of many thousands of binary digits (called bits) each of which is capable of taking one or other of the two values Ø and 1. These bits are grouped together to form 'cells'*; each cell has its own address by which it is known:



The store is used to hold information, all of which must be coded in some way so that it can be expressed in Ø's and 1's.

It is fairly easy to see that numbers like 134 can be expressed in binary so that

ØØØØ ØØØ1ØØØØ11Ø

could represent 134

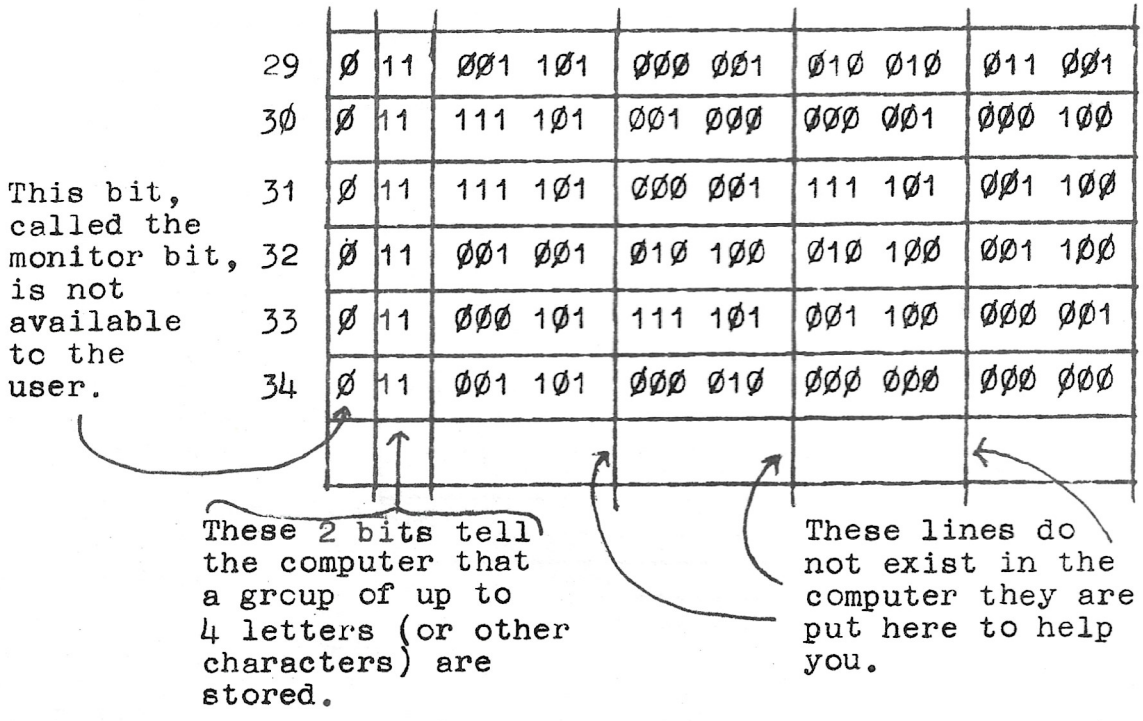
* Sometimes called 'registers' or 'store locations'

but much of the information that a computer handles is not easily thought of as a series of positive integers so that different methods are used for storing different kinds of information.

One of the chief kinds of information which a computer store must hold is the set of machine code instructions which make up a program. Most of the time, a computer works by obeying instructions which are already in store. This implies that two stages are involved if we want the computer to do a particular job. Stage 1 consists of putting the instructions to be obeyed in store and stage 2 consists of obeying them.

Examples 1

1.1 One way of coding information which can easily be expresse in alphabetic form is to code each letter into binary. In the BBCX computer it is possible to pack 4 letters (or other characters) into each cell. Each character is represented by 6 bits and the code used for this is fairly obvious. What message has been stored here in cells 29 to 34?
(It might help to know that 111 1Ø1 represents a space).



1.2 Using 6 bits to code each character, how many different characters can be coded? (In this context, a space is regarded as a character, but ØØØ ØØØ is used to represent 'no character')

1.3 In 1.1 it was seen that the two of the three left hand bits (called the 'type' bits) were used to specify how the other 24 bits were to be regarded. How many different types can be described using 2 bits? What other types would you think it might be useful to have?

Chapter 2 The First Program

```
16  TAKE 19
17  ADD  2Ø
18  PUT  21
```

This (very short) program consists of three instructions which are to be stored in cells 16, 17 and 18. Some of the cells with addresses from Ø to 15 have special properties and are not usually used for storing instructions so that the first instruction in a program often goes into cell 16. One of the special cells (with address 1) is called an accumulator and it is this cell which is used to store one of the two numbers (or other store content) which the computer is to process. The instruction TAKE 19 means:

'Throw away the current content of the accumulator (cell 1) and replace it with a copy of the content of cell 19.'

ADD 2Ø

'Add a copy of the number in cell 2Ø to the number in the accumulator and leave the answer in the accumulator.'

This instruction implies that the current content of cell 2Ø and the current content of the accumulator are both to be considered as numbers (for the moment, whole numbers).

PUT 21 means

'Throw away the current content of cell 21 and replace it with a copy of the content of the accumulator.'

It is important to notice that TAKE and PUT both mean 'copy'. The original, which is being copied, is left intact.

Notation In future we shall write 'C( )' for 'content of' 'Acc' for 'the accumulator' and ':=' for 'becomes a copy of'. Using this notation:

TAKE 19	means	C(Acc):=C(19)
ADD 2Ø	means	C(Acc):=C(Acc)+C(2Ø)
PUT 21	means	C(21) :=C(Acc)

The program in cell 16, 17 and 18 is not much use as it stands because so far we have not said anything about the content of cells 19 and 2Ø. Suppose we arrange for C(19) to be +153 and C(2Ø) to be +231. We could do this by writing the program as:

```
16  TAKE  19
17  ADD   2Ø
18  PUT   21
19  +153
2Ø  +231
```

This way, the numbers +153 and +231 are translated into binary by the assembler along with the instructions TAKE, ADD and PUT, but we still have difficulty because although the computer is