

*An E-A world exclusive project:*

# Build your own digital computer!

This is the first of a series of articles describing a complete general purpose stored-program digital computer. Designed specifically for home construction, the computer has an instruction repertoire which includes all of the basic instructions available on most modern minicomputers. Despite this the cost has been kept down to a level far lower than any commercial machine, and the construction made very straightforward by having virtually all the wiring on printed boards. These features should make it an ideal project for anyone keen both to learn in detail how a computer works, and to have a small computer with which to gain experience in programming:

by JAMIESON ROWE

A few years ago when I first became interested in digital electronics, one of the things I found was that there were precious few introductory books on the subject — and not many of these were both readable and satisfying. This made the going very tough, and it was because of my own difficulties in getting to grips with the subject that I subsequently wrote the series of articles which became our handbook "Introduction to Digital Electronics."

That book has been very popular, with more than 11,000 copies having been sold to date. It has become widely used as a text and reference in schools and technical colleges, and although it is now in need of updating I believe it has been of some value in helping others to gain an insight into the basic ideas involved in digital circuits and their operation.

There is one section of the book, however, that I am aware is likely to be found less satisfying than the rest: that dealing with the digital computer. While I believe this section goes a little further than is found in many basic works on the subject, it is still likely to leave the serious reader with a great many questions about the actual design and operation of a stored-program machine.

Basically I think this shortcoming reflected my own modest understanding of these matters, at the time the book was written. And to this day there are very few, if any, books or other sources from which one can get more than a superficial understanding of the real "nitty gritty" of computers.

Happily in my own case I was lucky enough to have the opportunity to gain first-

hand insight into computer operation and programming. Thanks to the generosity of one of our parent companies, John Fairfax and Sons Ltd, and the friendly co-operation of their EDP Managers and staff, I was able to spend many hundreds of lunch-time hours writing programs and running them on their Digital Equipment PDP-8 minicomputers.

It was this extremely valuable experience which allowed me to finally grasp many of the points about computer operation which had never quite emerged from reading books. Truly, there is no real substitute for "hands on" experience, at least as far as the basic operation of computers is concerned!

However there was still one whole area into which even this experience had not provided insight, even though it had laid a very worthwhile foundation: the actual electrical design of a computer. And being an incurably inquisitive person, this naturally became the most intriguing aspect of all.

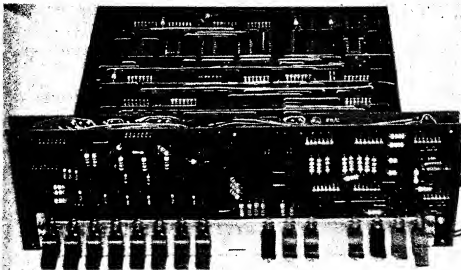
If, like myself, you've ever tried to get hold of the circuits of a minicomputer, with a view to finding out just how they function in fine detail, you'll probably agree that this is a well-nigh impossible task. I have never found such circuits in any textbook, nor have I been able to find them in any of the voluminous manuals which seem to accompany the machines themselves. I can only assume that the manufacturers prefer to keep the exact details to themselves and their servicing people.

There seemed to be no option, then, but to sit down with some modern digital IC data books, and design my own small computer based purely on a knowledge of general operating principles.

This, then, is the story behind the project itself. It has been a very interesting and challenging one, and I believe that it has taught me a great deal about the general principles of small computer design. I hope that at least some of these rewards may be passed on to readers, as a result of its description. I have spent considerable effort in trying to ensure that construction of the unit itself will present as few problems as possible, and will be doing my best in these articles to help the reader get the most benefit from the overall exercise.

But enough of this preamble. By now, the reader is no doubt wondering if I am ever going to get around to introducing the gadget itself!

Perhaps the best way to start is by describing the computer as a scaled-down version of the Digital Equipment PDP-8 machine — at least, in terms of its instruction repertoire and basic console layout. It has the same eight basic types of instruction, and for convenience I have used the same mnemonic names and similar coding.



A view of the "works" of the computer, taken before it was fitted into the case. Virtually all of the wiring is taken care of by the PC boards.

This has not been done purely because of my own initial experience with the PDP-8 family of machines, nor out of a simple desire to flatter the Digital Equipment people — although they have undoubtedly played, and are still playing, a leading role in developing minicomputer technology. While the design no doubt reflects in good measure my familiarity with these machines, and I am happy to pay tribute to the eminence of DEC, this superficial plagiarism has been done for a far more down-to-earth reason.

The fact is that Digital Equipment has led in the minicomputer field right from the start, and there are many more DEC machines in use than any other. Therefore the chances are great that if the reader, after building my little computer and gaining experience with it, has the opportunity to graduate to a commercial machine, it is likely to be a DEC machine. By making the operation of my machine as similar to a PDP-8 as possible, I hope that this transition to bigger and better things will be made easier.

I should perhaps add that most minicomputers have a great deal in common, so the fact that our machine is styled on the PDP-8 should not be seen as making it too specialised. In fact if the reader builds up this machine and uses it to get a solid grounding in basic computer operation and programming, he or she should find very little difficulty in making the transition to almost any brand of commercial minicomputer.

Essentially, then, this computer is rather like a PDP-8 and other modern minicomputers, except that it is a good deal smaller. Not so much in terms of physical size, because the latest generation of minicomputers are quite small themselves. The contrast is more in terms of circuit complexity, which has been scaled down both to reduce the cost and to bring the whole size of the project within the scope of a single individual.

Probably the most obvious difference is in terms of memory size, and this is because the memory is the most costly part of any computer. Whereas most modern minicomputers have a memory with at least a thousand or so storage locations, the memory of this machine has only a modest thirty-two.

While this much smaller memory capacity would make the machine of very limited use in practical computing applications, don't assume from this that it is purely a toy. A memory of 32 locations is entirely adequate for demonstrating virtually all aspects of basic computer operation and programming.

In fact, it is surprising just how complex and powerful a program one can fit into a memory this size, as I have been discovering! And in a very real sense, the small size of the machine's memory makes programming that much more of a challenge. It is relatively easy to write a program to perform a certain task, as any experienced programmer will tell you — the real trick is to achieve this end as economically as possible.

With a memory of only 32 locations, one cannot help being steered automatically in the direction of economic programming!

This is not to say, of course, that a larger memory would not have been desirable. A larger memory would certainly broaden the practical scope of the machine, and when the description of the basic machine and its

## About this unique project . . .

Our description of this project is, we believe, a unique and historical event in the development of popular electronics journalism. To the best of our knowledge it is the first time, anywhere in the world, that a complete general purpose stored-program digital computer has ever been described for home construction.

We believe it is particularly fitting that Electronics Australia is the first magazine to publish such a project. As one of the longest established electronics magazines in the world, it has always played a leading role in the do-it-yourself project and learn-while-you-build tutorial fields, and this project marks a milestone in both these areas.

We are also proud that the project has been developed wholly by Editor Jim Rowe, a member of the E-A staff for nearly fifteen years. Although the project has been developed almost entirely in his own time as a "labour of love", it continues the tradition which we have carefully built up over many years: that of describing as many projects as possible which have been developed by our own staff, thoroughly tested, and which therefore may be tackled by readers with confidence.

We believe this policy has played an important role in establishing E-A as the leading electronics magazine in Australia, and one of the leading magazines in the world.

This project will, we believe, have a particularly strong appeal not only for individuals, but also for high schools, technical colleges and other training establishments. This is because it provides both a means of learning in detail the exact operation of a modern stored-program digital computer, and also the opportunity (when the computer is completed) of gaining direct experience of computer operation and basic programming.

There have been many books published which cover the general field of digital electronics, our own "Introduction to Digital Electronics" included. Similarly there have been numerous logic training devices described, like our own Logic Trainer of March and April 1973. However apart from giving the enthusiast a general idea of the basic modules used in digital computers — such as shift registers, binary adders, and logic gates, these books and devices generally reveal little of how a digital computer is actually put together, and the detail of how it works.

To be sure, one is told that these basic modules are used to make up the broad sections of a computer, such as the "arithmetic unit", the "control unit" and the "memory unit". But little has been said of just what these sections comprise, and exactly how they interact with one another to produce an operating stored-program machine.

It was because Jim Rowe himself found these intriguing "fine details" of computer design and operation inadequately explained, that he decided many months ago to tackle the design and construction of a small computer for himself. He believed that by doing this, he would not only learn these elusive secrets for himself, but would also as a result be in the ideal position to pass them on to others.

In addition, there has been the second strong motivation for describing a low cost do-it yourself computer: the fact that, for the first time, this will enable almost anyone to sit down with a computer and obtain the "hands-on" experience which is so essential for a full understanding of how these revolutionary machines work.

As anyone who has ever played with a computer will tell you, there is absolutely no substitute for this sort of experience. Until one has actually played with a computer, these machines will always remain partly clouded by an element of awe and disbelief.

Modern minicomputers are very much lower in cost than the first computers, but they are still rather too costly to make them sufficiently accessible to all who really need this "hands-on" experience: the private individual, the school pupil, and the college student.

It is our firm belief that Jim Rowe's little "EDUC-8" computer will play an important part in bringing computer technology down to these very people. For it is not only a do-it-yourself construction project but a means of providing a small, yet fully operational computer within the reach of almost everyone's pocket.

peripheral devices is completed, I plan to look at the possibility of enlarging the machine in this way. But unless memory devices fall dramatically in cost, this will probably involve a significant increase in the overall cost of the machine, so that for the present I must ask the reader to accept my assurance that the present memory size represents an entirely acceptable place to start.

Apart from the memory size, there are two other main ways in which the computer is scaled-down in comparison with a typical commercial minicomputer. One is the word length, or the size of the binary numbers used in the machine to represent both instructions and data numbers. Whereas most commercial minicomputers have a word length of either 12 or 16 binary digits or "bits", in this case the word length is only 8 bits.

Again, this has been done to reduce both cost and circuit complexity. The main

penalty arising from the reduced word length is that the machine can normally deal with only modest numbers — between plus 127 and minus 128, inclusive. The relatively short word length has also made it necessary to reduce the number of augmented operate instructions, as will be explained later. I believe both these limitations are acceptable in view of the savings made possible.

The other main way in which the machine has been scaled down is in terms of the number of input and output devices or "peripherals" which it is designed to communicate with at any one time. Whereas many modern minicomputers are designed to "interface" with up to as many as 32 or 64 input and output devices at any one time, this machine will only interface with up to two of each — four in all.

This is a very minor limitation in a machine designed mainly as an educational tool, I believe, because the need to deal with large

## DIGITAL COMPUTER

numbers of peripheral devices is largely confined to practical data logging applications. Just about all of the basic principles involved in input-output transfer can be demonstrated using two input and output devices, and in fact more devices would probably only confuse the situation. In any case, the machine has been arranged so that the input and output devices simply connect to it via sockets on the case rear. If a total of more than four peripheral devices are built up or acquired, it is therefore quite easy to substitute devices as needed. Only four devices may be plugged in at any one time, that is all.

What peripheral devices may be used with the computer? Within reason, almost anything. As part of the basic description, I hope to describe a suitably simple and low cost input keyboard unit, together with one or two appropriate output display units. After this I hope to give details for interfacing with punched-paper tape readers and punches. It may also be possible to give details of other types of input and output devices, if opportunity permits.

To a large extent, the range of possible input and output devices to which the computer may be connected will be limited mainly by your imagination. There seems no fundamental reason why it could not be hooked up to all sorts of equipment, for all sorts of jobs. I myself aim to try out a few interesting possibilities, and I hope that readers will be encouraged to do the same.

Of course, in view of the modest memory capacity of the basic machine, it will probably be necessary in some cases to come up with some pretty fancy interfacing circuitry to achieve the desired result. In other words, to make use of some relatively high-powered "hardware" in the peripherals to make up for the fact that the computer itself can't accommodate much "software" — ie, much program.

Still, if you're at all adventurous, this should simply add to the challenge. My main aim in these articles will be to help you gain the necessary expertise, so that you can wherever your imagination leads.

At this stage, I should perhaps devote a few words to explain the basic electrical and physical construction of the computer.

All of the main circuitry of the machine is designed around TTL digital integrated circuits (ICs). There are a number of medium-scale or MSI devices, used mainly for the various registers, and one large-scale or LSI device — the memory IC. Apart from these, the remaining devices are all low cost "garden variety" 7400 devices. All should be readily available.

No doubt some will wonder why I have used not one of the new "microprocessor" LSI devices as the heart of the machine. This would have reduced the number of ICs used, to be sure; in fact the whole computer would probably have fitted on a single small printed board. However I decided against this approach quite early in the development of the project, for two reasons which I believe are conclusive.

The first is cost. Microprocessor chips are still quite costly at the time of writing they sell for about \$110 each. By the time one adds a memory device and the additional circuitry required to interface with a control console and input-output devices, this would make the cost of the resulting

machine considerably higher than the one presented here. And I don't know about the reader, but I myself would probably be far too nervous to play around with any single IC costing more than a hundred dollars!

But quite apart from the cost, there is the matter of teaching potential. A microprocessor chip is ideal for manufacturers wanting to build a computer into their systems as an integral part, but by its very nature it is not at all suitable for learning how a computer functions. With virtually all of the computer hidden away inside a 40-pin IC package, all of the real mysteries remain the property of the IC maker.

As it stands, there are a total of 96 ICs in the basic computer, most of these being low cost 7400 series devices. Some 86 are used in the basic processor and memory sections, while the remaining 10 are used in the internal input-output interface circuitry.

If this number seems a bit large, consider that the ICs probably incorporate something like 7,000 identifiable transistor elements. Small wonder that this computer project just wouldn't have been worth considering before MSI and LSI devices became a reality about two years ago.

Even with only 96 ICs to worry about, of course, the computer is still not a small project by do-it-yourself standards. If it had to be wired up in the old fashioned way, it would be an enormous job, and I doubt whether anyone would have been even remotely interested.

With this in mind I have designed the unit so that virtually all of the wiring is performed by printed boards. There are eight boards in all making up the basic computer: a board which takes care of all the front panel or "console" wiring, an interconnection or "mother" board, and six plug-in boards which make up the various functional parts of the machine.

As the mother board performs virtually all of the interconnections between the sockets for the plug-in boards, the actual hard wiring involved in building the computer is very small. Essentially it amounts to little more than making the few dozen interconnections between the mother and front panel boards, together with the power supply and input-output connector wiring.

I should mention here that to keep the cost down, I have used only single-sided printed boards for the project. This inevitably means that wiring-up each board involves fitting a number of wire links, in addition to fitting the IC's and the few other components. While this approach would not be appropriate for commercial manufacture, where the time needed to fit links might well prove more expensive than double-sided

boards, I believe it is the right choice where a home construction project is concerned.

The saving in board cost is quite dramatic, and for the enthusiast should far outweigh the small increase in assembly time and tedium. At least, I hope so.

The complete computer is housed in a metal case measuring about 30 x 10 x 35cm (WxHxD), or about 1 1/2 x 4 x 14 inches. The basic machine draws about 3 amps at 5 volts DC, or about 15 watts. However I have designed the power supply so that it can supply power to a number of peripherals, to simplify interfacing. As a result the total power consumption may in practice rise as high as 60 watts or so.

When one considers that this is probably still going to be less than the power consumed by the light bulb in the room you are using the machine, it provides dramatic evidence of the advances which have been made in computer technology over the last few years. The first computers were probably somewhat simpler than this little machine, yet occupied a complete room and needed their own power sub-station!

For the benefit of those who would like to know a little bit more about the computer they would end up with if they embark on this project, I have prepared a brief specification which is shown in the box panel. At this stage the specification may not mean much to you, however, because it really presupposes a knowledge of basic computer operation. For most readers, gaining this very knowledge will be the whole point of the exercise, and if they knew now they wouldn't be bothering.

The main idea behind giving it here is to show that the computer is quite a capable little machine, despite the fact that it has been scaled down for home construction. It has just about all the basic capabilities found in modern computers, and is therefore able to demonstrate most aspects of computer operation, although for the present you'll probably have to take my word for this.

Incidentally, when the basic machine has been described, I will then spend some time discussing programming. So you need not be concerned that having built the computer, you won't know what to do with it. Naturally you won't become an expert programmer, but I hope you will emerge from all this with a sound basic grounding — if you can stick with it.

Two aspects of the project remain to be discussed in this introduction, I think. One is the name we have given the computer. Like all good computers, it must have an impressive-sounding name — that's traditional. In this case we came up with the name "EDUC-8", and our excuse for this

**CHAPMAN'S  
MIDGET**

with Splinter  
Top adapters

**RATCHET KITS**

LOCAL SUPPLIES AND  
CATALOGUES AVAILABLE FROM

Local supplies  
and catalogues  
available from

**SULCO**

469 PACIFIC HIGHWAY, ARTARMON, N.S.W.  
Telephone: 42 4214

SPECIALISTS IN PRECISION FASTENING TOOLS

## SPECIFICATION OF THE COMPUTER

The machine is a complete stored program general purpose digital computer. Memory storage is provided by means of a bipolar static random-access integrated circuit (RAM), organised to have a capacity of 32 locations.

Both instructions and data words use a format of 8 bits.

Almost all processing is performed serially, with alternative clock rates of approximately 500kHz and 2Hz. Instruction cycle times are constant at either speed, at approximately 96 microseconds and 24 seconds respectively. Operation may be either continuous or on a single instruction basis, as desired.

There are five main registers, comprising the program counter (PC), the accumulator (AC), the memory address register (MA), the memory buffer register (MB) and the instruction register (IR). The front panel of the machine is also provided with a switch register for manual loading of addresses, instructions and data.

Six types of memory reference instruction are provided, consisting of logical AND (AND), binary addition (TAD), increment and skip if zero (ISZ), deposit and clear accumulator (DCA), unconditional jump (JMP) and jump to subroutine (JMS). A single address format is used for these instructions, but both direct and indirect addressing are available.

Eight augmented operate (OPR) instructions are provided, comprising increment accumulator (IAC), complement accumulator (CMA), clear accumulator (CLA), rotate accumulator 1 bit right (RAR), rotate accumulator 1 bit left (RAL), skip on zero accumulator (SZA), skip on minus accumulator (SMA) and halt (HLT). A number of these instructions may be combined, for programming economy.

Three augmented input-output transfer (IOT) instructions are provided, each of which may be arranged to specify either an input or output device, and one of two possible devices in each category. The instructions are skip on device flag (SKF, SDF), transfer data between device buffer and accumulator (KRS, LDS), and reset device flag (RKF, RDF). The second and last of these may be combined for programming economy.

The contents of all registers are displayed on the front of the machine by means of light-emitting diodes (LEDs), allowing the operation to be readily followed.

Power consumption of the basic machine is approximately 15 watts, but rises to approximately 60 watts with some combinations of input-output devices.

somewhat transparent pun is that the first part stands for "Educational Digital micro (u) Computer", while the figure 8 represents either the number of basic instruction types, or the word length, or both!

Anyway, it IS a small or "micro" computer, and we hope it will prove useful as an educational tool. It must have some sort of name, and EDUC-8 is probably as good as any.

Finally, there is the matter of cost. You have probably been wondering if I was ever going to be specific about this, and the truth is that I have deliberately left it until now to give you a chance to put the project in perspective.

It is very difficult to be accurate about cost, because as most readers will be well aware, prices of electronic components are fluctuating all the time. About the best I can do is tell you that as far as we can estimate, the complete basic computer will probably cost you about \$300 — maybe a little more, quite possibly a bit less.

Don't forget that because the project is built on a number of boards, you can build it up progressively a module at a time, and distribute the outlay over a period of time. We will be describing the project with this approach specifically in mind, in fact, as quite apart from the cost angle there are other advantages in tackling the project in stages.

With a project this size there is a lot to be said for building up the key sections one at a time, and checking them out individually before the final assembly as a system.

Well, that's the basic story on this new project of ours. I hope some of my own enthusiasm for it has rubbed off, and that by this stage some of you will be champing at the bit.

Before I close this introduction, I must give grateful thanks to the many people who have helped me very considerably in the development of the project. I am indebted to

John Houston and Val Rech of Fairchild Australia, Ron Bell of RCS Radio, Doug Evans of Ferguson Transformers, Peter Carter of A & R Soanar, David Segal of Philips Elcoma, Brian Cleaves of Plessey Australia, Norm Volkman of Wardrobe and Carroll Fabrications, and Erle Goodwin of General Electronics Services — to name but a few of the many who generously contributed both advice and samples. I hope that those not named in the foregoing will please assume, rightly, that this is purely because of lack of space.

There are also my fellow staff members at E-A, who have been very tolerant of my involvement with and enthusiasm for the project. I must particularly express thanks to Bob Flynn, who has been of great help in the preparation of the printed board patterns.

And last, but by no means least, I must express my very grateful thanks to my wife Laraine, for putting up with the long lonely nights and weekends when I have been either submerged in the workshop, or present physically but almost constantly preoccupied with a logic problem or printed board pattern. The things we electronics addicts inflict on our loved ones!

## STOP PRESS!

Just as this issue was going to press, an advance copy of the July issue of the US magazine "Radio-Electronics" reached us. In it they give details of Mark-8, a microcomputer based around the Intel 8008 microprocessor IC and 1101 memory ICs. Our design is not the first, then, as it transpires — we were beaten by a few weeks!

Ah well, such is life. Still, readers will now have a choice of two computer designs, each with rather different emphasis.

# Know Where You are Going

Choose a career in the field of Electronics — the Nation's most progressive and fastest expanding industry.

Advancement in this modern science demands technical ability, a sound knowledge of basic principles and applications. You can master the subject by training at the Marconi School, and be ready to grasp the opportunities that occur in the various branches of Radio Technology.

## BROADCASTING

A thorough grounding is available to students in the broadcasting field, leading up to the P.M.C. Broadcast Station Examination.

## COMMUNICATIONS (Marine)

An extension to the Broadcast Operators course and extending into the fields of Navigation Aids and Electronic Devices used in mobile communications as required by the P.M.G. Certificates of Proficiency.

## APPLIED SERVICING

Comprehensive training in the maintenance and repair of radio and television receivers offers substantial rewards to competent technicians. Marconi School training covers all aspects of radio and television receiver circuit applications, practical exercises in fault finding and alignment procedures.

The Marconi School Radio Servicing correspondence course is approved by the N.S.W. Apprenticeship Commission

Classes are conducted at:  
67 Lord's Road, Leichhardt.

Day: 9 a.m. to 4 p.m.

Evenings: 6 p.m. to 8.30 p.m. or by Home-Study Courses (except practical instruction on equipment).

## SEND FOR PROSPECTUS

There is no obligation

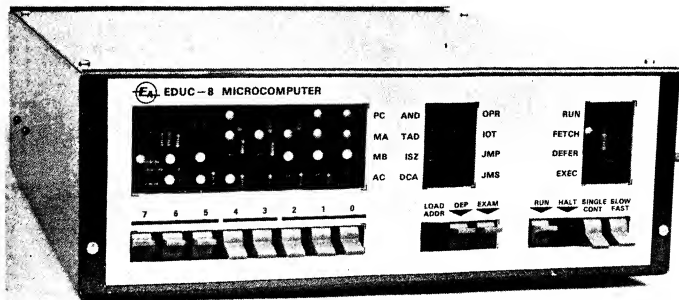
NAME.....

ADDRESS.....

## MARCONI SCHOOL OF WIRELESS

Box 218, P.O., Leichhardt 2040

A service of  
Integrated Wireless (Australia) Ltd.



# Our do-it-yourself computer: how it works

This is the second in a short series of articles describing EDUC-8, our uniquely conceived build-it-yourself digital micro-computer project. Leading on from the introduction given last month, the author describes the various sections and registers making up the machine, and shows how they operate together as an automatic system.

by JAMIESON ROWE

Before worrying about the construction of our microcomputer, no doubt most readers will want to make sure that they have a reasonably clear idea of the way it works. I am therefore devoting this article to a description of the basic operation of the machine.

At this stage the aim will not be to make the reader intimately aware of every fine detail of the computer's operation, but rather to paint in the broad picture. More detailed discussion of the operation of each section will be given, but later on as we deal with the construction. I believe this progressive approach will help the reader to assimilate the concepts more readily.

Let us begin with a brief revision of basic concepts. You will perhaps recall from previous reading that digital computers are rather unique electrical machines, in that they are capable of performing not just one function, but a number of alternative functions. In fact each computer has a repertoire of basic functional "tricks", any one of which it can perform upon command; ie in response to an appropriate instruction.

By making the tricks in its repertoire as

basic as possible, and by providing enough of them, the machine is made capable of performing an almost infinite variety of tasks. Any given task to be performed is analysed, and broken down into a logical sequence of the computer's basic tricks. It then follows that if the machine is made to follow the appropriate sequence of instructions, known as a program, it will perform the desired task.

By changing the program, the machine can be made to perform different tasks at will. But the point to grasp is that whatever task is to be performed, the appropriate sequence of instructions must be devised, and fed into the machine. Although rightly called a "general purpose" machine, a computer can't do any task at all if it is not provided with a program.

The important thing to remember about the instructions which command the machine to do its basic tricks is that they are binary numbers. Nothing more, nothing less. So that the program which tells the computer how to do a certain task is simply a string of binary numbers stored in its memory.

What is the difference between the in-

struction numbers stored in the computer's memory, and any other numbers stored there? There is no difference as such. It is all a matter of interpretation.

Essentially, any and every binary number stored in a computer's memory is potentially capable of making the machine perform one of its basic tricks. But at the same time, any given number in the memory only becomes an effective instruction when the computer interprets it as such. So that the trick is to ensure that the machine only interprets as instructions those numbers intended to be such.

Basically this is done by indicating the location or address in the memory of the first instruction of a program, before pressing the "run" button to set it in operation. In the normal course of events the computer then interprets the numbers in successive memory locations as the subsequent instructions, unless commanded to do otherwise by one of the instructions. Hence if the computer is started at the correct starting address, and the program is correct (!), only those numbers stored in the memory as the program will be interpreted as such.

Because its operation involves performing a sequence of instructions stored in its memory, a computer operates in a rhythmic or cyclic fashion. In operation it repeatedly fetches an instruction from the memory, interprets it, and then performs or executes the appropriate trick in its repertoire. This cyclic fetch-execute-fetch-execute... sequence continues until halted either by

the operator, or by an instruction which itself signifies "halt".

This basic sequence of operations is shown in Fig. 1, which is a simple example of what is known as a flow chart. Such diagrams are often used in analysing and describing computer operation, as we shall see later, because they make it very easy to visualise what is going on.

In this case the chart shows that after starting and performing the fetch-execute cycle for the first instruction, the computer effectively makes a logical decision as to whether it should halt or not. If the answer is no, it returns to fetch and execute a second instruction, and so on. This continues until the answer to the halt decision becomes "yes", either because the operator has pressed the halt button, or because the last instruction performed happened to be "halt".

With the foregoing hopefully clear in the reader's mind, let us now look at the main sections of a computer with the idea of seeing just how it performs its tricks. Perhaps the easiest way of doing this is to look first at the sections of the machine involved with instructions, and then at those involved with data. Some sections will turn out to be involved with both, but this should not be a problem; our next step will be to see how the two groups are fused together to form the complete machine.

Fig. 2 shows the basic sections of the machine involved with instructions. Probably the most easily recognised of these is the memory, and associated with this are two registers known respectively as the **memory address register (MA)** and the **memory buffer register (MB)**. A register, you may recall, is a set of flip-flops or similar storage devices capable of containing a binary number.

The function of the MA register is to hold and indicate to the memory the binary number address which specifies the memory location with which the machine is concerned at any particular instant. So that during the fetch phase of the machine's cycle, when the next instruction in the program must be read out of the memory, the MA must contain the correct address of that instruction.

This "next instruction address" is fed to the MA from the **program counter (PC)**, a register whose sole function is to keep track of the computer's progress during a program.

Initially, the starting address of the program is fed into the PC, for example by the operator using the console switches. Then, when the number is fed from the PC to the MA to allow the first instruction to be fetched, at the same time the number is incremented (increased by one) and fed back into the PC. This ensures that during the next fetch phase, the MA will be fed with the address of the second instruction. At that time, the number is again incremented, and fed back into the PC ready for the third fetch phase, and so on.

The function of the MB register is to act as an intermediary between the memory itself and the rest of the machine, as far as the numbers actually stored are concerned. Thus when an instruction number is read out of the memory during a fetch phase, it first enters the MB — to gain its composure, as it were.

From the MB register, the part of the instruction number known as the operation code is passed on to a further register, known as the **instruction register (IR)**. Don't worry about the significance of the

operation code now, as we will look at instruction coding shortly. For the present, it is sufficient to note that this part of the instruction number specifies which **type** of instruction has been fetched, and the function of the IR is to "remember" this vital information during the subsequent execute phase. Without the IR, the computer would literally forget what it was supposed to be doing!

Within the IR register, the operation code is still in the form of a binary number. Associated with the IR is therefore an instruction decoder, whose purpose is to interpret this number and generate the various logic gating signals necessary to perform the appropriate trick.

These, then, are the main sections of the computer involved with instructions. Now let us turn to look at the sections involved with data — i.e., the numbers with which the computer is to perform its tricks. These sections are shown in Fig. 3.

As you can see, three of the sections from Fig. 2 also appear in Fig. 3 — the memory, the memory address register and the memory buffer register. And these perform the same functions with data as they did

with the functions involved in transferring data between the computer and its "peripherals" — the input and output devices. This involves not only the simple shuffling of numbers in one direction or the other, but associated tasks such as letting one side know when the other is ready to transmit or receive.

At this stage we have looked briefly at those sections of the computer which are involved with handling either instruction numbers, or data numbers, or both. However the thoughtful reader already may have begun to suspect that there are still further sections of the machine, whose functions have as yet only been implied.

These are the sections involved with run control and timing, and they are shown in basic form in Fig. 4.

Basically the need for these sections arises because the computer operates in a cyclic fashion, as we have seen. Not only this, but in fact both fetching an instruction and executing it each involves a number of steps, which must be done in sequence. Gates must be opened and closed, registers fed with control signals and clock pulses, and so on, all in the right order. And quite apart from this there is the need to be able to control whether the machine is running or not, and similar control functions.

The sections which perform these tasks are the clock oscillator, the run control circuit, the console control switches, the timing pulse generator, and the major state generator.

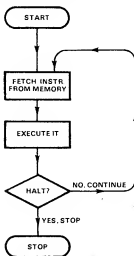
The purpose of the clock oscillator is to generate the master pulse train which activates the entire machine when it is running. It is therefore by controlling the passage of these clock pulses to the remaining sections that the run control circuit functions. Broadly speaking, it lets the pulses pass following a command to do so by the console control switches, and continues to let them pass until a command to halt is received — either from the switches, as before, or upon the arrival of a halt instruction.

When the computer is running the clock pulses pass from the run control circuit to the next section, the timing pulse generator. The function of this section is to use the master clock pulse train to assemble a number of timing and gating signals, each of which may be used in various places throughout the machine to perform operations at certain times in each cycle.

Finally there is the **major state generator**, whose function is to define which of the phases or "major states" the machine is in at any given time. Immediately upon being set running, the machine always enters a fetch state or cycle, as we have seen from Fig. 1. At the end of the fetch state it then normally enters the execute state, although as indicated in Fig. 4 there is a third and intermediate state known as "defer". More will be said about this shortly. Finally at the end of the execute state, the machine either returns to fetch and continues, or halts.

I hope the foregoing description has at least given the reader a broad idea of the basic sections which form the computer, and of the general functions they perform. At this stage there will still be many questions unanswered, and you will perhaps be feeling rather unsatisfied. However now that some of the foundations have been laid, we can delve into things in a little more detail.

Let us first look at the instruction coding format used — i.e., the way in which the binary numbers which act as the in-



1: BASIC OPERATION

with instructions. The MA register is used to specify the address of a memory location into or from which a data number is to be transferred, while the MB register is used as an intermediary store between that location and the remaining sections of the computer.

Together with these three sections we now have an accumulator register (AC), an arithmetic and logic section, and an input-output interfacing section.

Essentially the function of the accumulator register is to retain or accumulate the results of calculations, as they are being performed. If you like, it is the "blackboard" which shows how the sums are progressing.

The arithmetic and logic operations are not generally done in the accumulator itself, but in the associated arithmetic and logic section. However this section, the AC register and the MB register are closely associated, and all three are to some extent involved in most calculations. In binary addition, for example, the two numbers being added are placed initially in the MB and AC registers, and after they are added together by the arithmetic and logic section, the answer is placed in the AC register.

The final main section concerned with data is the input-output interfacing section. As the name suggests, this section carries

## EDUC-8 computer

structions for the machine are actually arranged to specify what is to be done.

As you might recall, EDUC-8 uses an 8-bit word format. In other words, both the instruction and data numbers handled in the machine are 8 bits long. For convenience the 8 bits are labelled from 0 to 7, with bit 0 being the least significant bit in terms of binary weighting, and bit 7 the most significant.

When such a number is interpreted by the computer as an instruction, the three most significant bits — bits 7, 6 and 5 — are taken to represent the **operation code**. It is these three bits which are stored in the instruction register and decoded, to specify which type of instruction "trick" is to be performed.

Now you will perhaps remember from basic theory that three bits of information can only represent eight different situations, because there are only eight different bit combinations: 000, 001, 010, 011, 100, 101, 110, and 111. This means that there are basically only eight different types of instruction in the machine's repertoire.

Six of these eight basic types of instruction are known as **memory reference instructions**, because they each involve some sort of operation with a number stored in a location of the memory. Either a number is taken out of the location and used for an arithmetic or logic operation, or a new number is stored in the location, or the number is interpreted as the machine's next instruction, and so on.

The combinations of the three operation code bits used to represent these six instructions are 000, 001, 010, 011, 100 and 101. For convenience we will refer to the equivalents of these in the octal code, which are 0, 1, 2, 3, 4 and 5. It is very much more convenient to think in terms of octal code than in binary, and I suggest you brush up on the simple relationship between the two. We will be using octal notation fairly frequently from now on, but purely because we humans find it easier to follow. The computer itself deals purely with the numbers in binary form, of course.)

The coding format used for memory reference instructions is shown in Fig. 5(a), where it is indicated that the operation code bits 7, 6 and 5 here specify one of the six combinations just referred to. The four least significant bits (0-3 inclusive) are used to define the address of the memory location whose contents are involved in the operation concerned. This is known as the **operand address**.

Don't worry for the moment about the special significance of bit 4; we will return to this shortly. Our more immediate concern is the six types of memory reference instruction, and what each involves. Their mnemonic names are probably already vaguely familiar, because they were given in the specifications panel. Hopefully now you will be in a position to follow the operations themselves.

The instruction type with operation code 0 (octal) is known as the **AND** instruction. Basically this involves reading out of the memory the number stored in the location specified by bits 0 to 3, and performing a logical **AND** operation between this number and the number currently in the accumulator register. The **AND** operation is done on a bit-by-bit basis, and the result placed in the accumulator. As the accumulator will be left with binary 1's only in those bit positions

where both initial numbers were 1's, the effect is to "mask" the number in the accumulator with that in the selected memory location. As such, it can be a very useful operation.

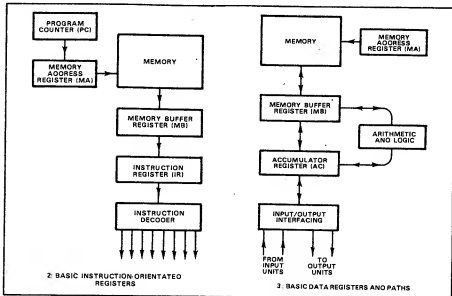
The instruction type with operation code 1 (octal) is known as the **TAD** instruction, which is short for "Two's addition." This is rather like the **AND** instruction, only the operation performed between the number read out of the selected memory location and that in the accumulator is binary addition, using 2's complement arithmetic, instead of **ANDING**. As before the result is left in the accumulator. The **TAD** instruction is the main arithmetic instruction; it can be used for subtraction as well as addition, by forming the 2's complement of the number in the accumulator before the **TAD** operation is performed.

The instruction type with operation code 3 (octal) is known as the **DCA** instruction, short for "deposit and clear accumulator." As this suggests, it simply involves taking

stored again in the same memory location. But if the number has become zero after being incremented, the machine also increments the contents of the program counter. This means that the next instruction fetched will not be that in the next consecutive location, but that in the one after that. In other words, the machine "skips" what would normally be the next instruction.

Although it may not be apparent at this stage, the **ISZ** instruction is a very powerful one. Basically, it allows the machine to make changes in the sequence of instructions automatically during a program, as the result of checking its progress.

The final memory reference instruction is that with the octal operation code 4. This is known as the **JMS** instruction, short for "jump to subroutine." Like the **JMP** instruction, it involves replacing the contents of the program counter register, so that the next instruction is taken from somewhere other than the next consecutive location. But in this case the existing contents of the PC are



the number currently in the accumulator, and storing it in the memory location specified by bits 0 to 3. The accumulator is left cleared — i.e., with a content of zero.

The instruction type with operation code 5 (octal) is known as the **JMP** instruction, short for "jump." The effect of this instruction is to cause the address shown by bits 0-3 to be transferred to the program counter (PC) register, replacing the existing content of the PC. This means that instead of the machine fetching its next instruction from the next consecutive location, it will fetch it from the location specified in the **JMP** instruction. This is a very useful instruction because among other things it makes it possible for the machine to be forced to repeat a sequence of instructions many times.

You will probably have noticed that we have so far ignored the instructions with octal operation codes 2 and 4. These are a little more complicated than the others, so they should be studied a little more carefully:

The type 2 instruction is known as the **ISZ** instruction, short for "increment and skip if zero." It involves the following operations. First, the number in the memory location specified by bits 0-3 is read out, and incremented (increased by one). Then the machine tests the number, to see if it has become zero or not. The number is then

not lost, as they are with the **JMP** instruction; instead they are stored, so that in due course the machine can return to the next consecutive location and continue.

A subroutine, you may recall, is a small group of instructions which are used repeatedly throughout a program. Rather than simply repeat them at every place they are needed, which would gobble up memory space, it is far more efficient to store them only once, and simply arrange for the machine to jump over and perform them whenever they are needed. Naturally when this is done, it is essential that the computer be able to keep track of where it has come from in the main program, so that it can return. This is the reason for the **JMS** instruction.

Basically what happens during the **JMS** instruction is that the current contents of the PC are taken and stored in the location specified by bits 0-3 of the instruction. At the same time this operand address is incremented, and placed in the PC. This has the effect that the next instruction fetched is taken from the next consecutive location from that in which the original PC contents have been stored. In other words, the computer stores the address in the main program to which it will return, in the first location of a subroutine, and will then proceed to work through the subroutine.

## EDUC-8 computer

How it actually uses the stored return address to "get back" will be explained shortly.

Let us now return to Fig. 5(a), and look at the significance of bit 4. As you can see, when this bit is zero it implies something called "direct addressing," while if it is a one it implies something else called "indirect addressing."

Thus far in discussing memory reference instructions, we have tacitly assumed that the only possible significance of bits 0-3 was to specify the actual address of the location in the memory occupied by the operand of the instruction. However this is only one way of using bits 0-3, the way known as **direct addressing**. It is the simplest way, and perhaps the way most readily visualised.

However it is not the only way, nor the only desirable way. In fact there is another way of using bits 0-3, which adds very significantly to the flexibility of a computer from the programmer's viewpoint. In this alternative approach, bits 0-3 are interpreted as specifying not the actual operand address, but the address of a further location in memory, in which the actual operand address in itself stored. Not surprisingly, this approach is known as **indirect addressing**.

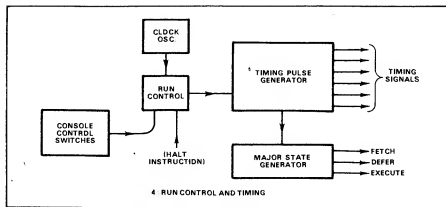
At first sight, indirect addressing may seem nothing more than a more complicated way of doing the same thing as direct addressing. But in fact it opens up all sorts of new possibilities. For one thing, it provides the means whereby the machine can "get back" to the place where it left a main program, from a subroutine. All that need be done at the end of the subroutine is to provide a **JMP** instruction with indirect addressing, which specifies the address at the start of the subroutine in which the original return address was stored during the **JMS** instruction.

Another very useful application of indirect addressing is where an operation must be repeated with the numbers stored in a whole string of consecutive memory locations. By using a number in another location as an indirect addressing "pointer," the numbers in the consecutive locations can be made accessible very easily, simply by incrementing the number in the pointer location, between operations.

Naturally enough, indirect addressing involves extra steps compared with direct addressing. The address specified by bits 0-3 of the instruction word must be fed to the MA register, and the actual operand address read out of that location.

The need for these extra steps is the reason behind the third major machine state, **defer**, which we referred to briefly when dealing with Fig. 4. In short, the machine only enters the defer state, between fetch and execute, if it is necessary to perform the extra steps necessary for indirect addressing. Indirect address memory reference instructions thus involve three machine cycles, whereas direct address instructions involve only two.

Indirect addressing also has an advantage over direct addressing in terms of access to all possible memory locations. Unless a rather long instruction word format is used, it is not possible to directly address all locations in the memory. This is because the full range of memory locations involves more addresses than there are bit combinations available in the instruction word, after the operation code bits and direct/indirect indicator bit are accounted for.



In EDUC-8, for example, only bits 0-3 inclusive are available for the operand address, and these four bits are capable of specifying only sixteen possible addresses, whereas the memory actually has twice that number. (If bit 4 were added, all 32 locations could be directly addressed, but we would then have no way of indicating indirect addressing.) A similar situation exists with many computers, particularly those using relatively short words.

This difficulty is avoided by arbitrarily splitting up the memory into groups of addresses called "pages," and adopting the convention that an instruction can only address directly those memory locations in the same "page" as the instruction itself. Thus in the case of EDUC-8 the memory is visualised as being split into two pages, consisting of the first sixteen locations and the last sixteen respectively.

The computer is then designed so that when direct addressing is involved, it automatically adds the missing fifth (most significant) bit of the operand address, making it the same as that of the instruction itself. In effect, it "assumes" that the specified direct address is in the same "page" as the instruction.

While with indirect addressing it is still necessary for the location in which the actual operand address is stored to be in the same page as the instruction, the operand address itself can occupy a full word, and can thus specify any location in the whole of

the memory. Thus with EDUC-8 the stored operand address can be anything up to 8 bits long, sufficient to specify 256 locations — ample for the 32 locations actually present, and with room for future memory expansion!

Let us now look at the two remaining types of instruction provided by EDUC-8, and the coding formats used for them. The first we will look at is that with the operation code 7 (local, which is known as the **OPR** or "operate" type of instruction.

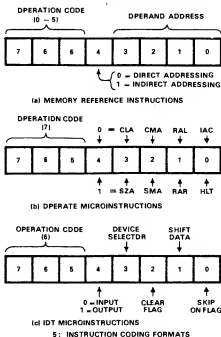
Although there is only the one operation code associated with this type of instruction, it is actually subdivided into eight different and distinct microinstructions, some of which may be combined to form further microinstructions. What makes this "subdivision" possible is the availability of bits 0-4; because this type of instruction does not make reference to the memory, these bits are not required for specifying an operand address. They can therefore be used to extend or "augment" the basic operation code.

The way the additional bits are used to specify the eight different **OPR** microinstructions is shown in Fig. 5(b). Rather like the memory reference instructions, bit 4 is used as an indicator bit. However in this case it is merely used to indicate which of two possible meanings is to be placed on 1's in the remaining four places. When bit 4 is zero, bits 0, 1, 2 and 3 are used to specify the four microinstructions **IAC**, **RAL**, **CMA** and **CLA** respectively. On the other hand when bit 4 is a 1, they specify instead the microinstructions **HLT**, **RAR**, **SMA** and **SZA** respectively.

The mnemonic **IAC** stands for "increment accumulator." As the name suggests, this microinstruction simply involves incrementing or adding one to the number in the accumulator. If the AC is initially zero, then it will contain the binary number 1 following the **IAC** microinstruction.

The mnemonic **RAL** stands for "rotate accumulator left." This involves shifting the number in the AC one bit position to the left, so that the content of bit position 0 moves to position 1, 1 to 2, 2 to 3, and so on. The content of bit position 7, if there is any, is shifted around into bit position 0. For binary numbers less than the equivalent of decimal 127, **RAL** corresponds to multiplication by two.

The third of this group of **OPR** microinstructions is **CMA**, which stands for "complement accumulator." It involves taking the number in the accumulator and changing it into its 1's complement — i.e., each bit is turned into its complement. Bits which are initially zero become 1's, and those which are initially one become zero. If the AC is initially cleared (all zeroes), the





## EDUC-8 computer

CMA instruction gives it a content of one.

The fourth OPR microinstruction is CLA, which stands for "clear accumulator." This is almost self-explanatory; the contents of the AC are simply wiped out, leaving the AC bits all zero.

The first of the OPR microinstructions for which bit 4 is a 1 is HLT, short for "halt." The effect of this instruction is simply to cause the run control circuitry of the machine to stop the passage of master clock pulses to the timing pulse generator, at the end of the current execute cycle.

The next of these is RAR, standing for "rotate accumulator right." This is virtually the opposite of the RAL microinstruction, causing the number in the AC to be shifted one position to the right. The content of bit 1 is moved into bit 0, that of bit 2 into bit 1, and so on. The content of bit 0 is moved around and into bit 7, so that as before no bits are lost. In effect the RAR instruction corresponds to a division of the number in the AC by two.

The seventh of the OPR microinstructions is SMA, which stands for "skip on minus accumulator." The effect of this instruction is to cause the content of bit 7 of the AC to be tested, to see if it is a 1 (by convention, the most significant bit of a number is taken to indicate its sign - 0 means positive, 1 negative). If this bit is a 1, the contents of the PC register is fetched not from the next successive memory location but from the location after that.

And lastly there is the SZA microin-

struction, which stands for "skip on zero accumulator." This is similar to the SMA instruction, but involves a test of all bits of the AC, not just bit 7. In this case the contents of the AC are incremented only if all AC bits are zero, so that a clear AC causes the next instruction to be skipped.

The SMA and SZA microinstructions are both very powerful ones from a programmer's viewpoint, because they provide a means whereby the program can test the result of a calculation, and automatically decide upon a course of action.

Some of the OPR microinstructions may be combined to form further microinstructions. Thus CLA and IAC may be combined, allowing the AC to be given a content of 1 by means of a single instruction. Similarly CLA and CMA may be combined, which has the effect of setting the AC to a content of -1. The CMA and IAC microinstructions may also be combined, allowing a number in the AC to be changed into its 2's complement using only one instruction. And finally the SZA and SMA microinstructions may be combined, so that the next instruction is skipped if the AC is either zero or minus.

The main advantage of these combined microinstructions is that they save memory space, allowing more program to be fitted into a given number of locations. They also make a program run faster, because in effect two separate "tricks" are done in a single execute cycle.

The final type of instruction provided by EDUC-8 is that with the operation code of 6 (octal). This is known as the IOT instruction group, where IOT stands for input-output transfer. Like the OPR instruction type, this

type is also sub-divided into a number of microinstructions, and as the name suggests, these are all concerned with the transfer of information between the computer itself and any input and/or output devices to which it may be connected. The coding format used for IOT microinstructions is shown in Fig. 5(c).

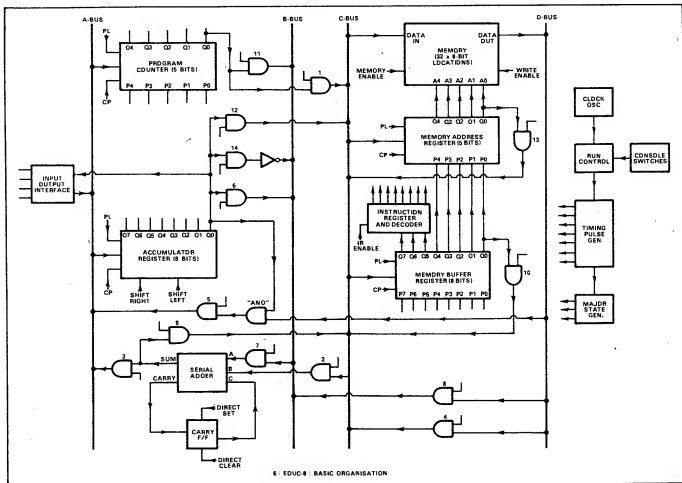
As before bits 7, 6 and 5 are used for the operation code, in this case 6 (or binary 110). And bit 4 is again used as an indicator bit, this time to specify whether the instruction concerned refers to an input device, or an output device. A zero is taken to specify an input device, and a 1 an output device.

Bit 3 is also used as an indicator bit, in this case to select which of the two devices of each type the instruction is concerned with. Thus if bit 3 is a 0, the instruction is concerned with either input device "0" or output device "0", depending upon the value of bit 4. Similarly if bit 3 is a 1, the instruction is concerned with either input device "1" or output device "1".

Between them bits 3 and 4 are thus used to specify which of the four possible input-output devices an IOT microinstruction is concerned with. Hence even when four devices are connected to the computer, each can be dealt with individually.

Bits 0, 1 and 2 of the IOT instruction format are used for the three different types of IOT microinstruction. In other words, these three bits form the augmented operation code.

When bit 0 is a 1, this corresponds to a "skip on device flag" microinstruction. The effect of this microinstruction is to check whether the input or output device concerned has signalled its readiness to take



## EDUC-8 computer

part in a transfer of data, by setting a flip-flop known as its "flag". If the flag is indeed set, the contents of the PC are incremented so that the next instruction is fetched not from the next successive memory location, but the one after.

There are two alternative mnemonics for this microinstruction, SKF and SDF, used for input and output devices respectively. In either case the purpose of the microinstruction is to allow the computer to determine when a peripheral device is ready to "do business."

The second type of IOT microinstruction is that having a 1 in the bit 1 position. This is the "shift data" microinstruction, represented by the mnemonics KRS (input) and LDS (output) respectively. As the name suggests, this microinstruction is the one which actually causes data to be transferred between the computer and a peripheral. In the case of an input unit, the data is transferred from the input unit into the AC of the computer. With an output unit, the data is transferred from the AC into the output unit.

Finally there is the third type of IOT microinstruction, in which bit 2 is a 1. This is the "reset device flag" microinstruction, represented by the mnemonics RKF (input) and RDF (output). Fairly obviously, this microinstruction causes the flag flip-flop in the device concerned to be reset, and it allows the computer to signal to the peripheral device that a data transfer is complete. This allows the device concerned to perform whatever housekeeping operations are needed to prepare for a further data transfer.

The "shift data" and "reset device flag" IOT microinstructions may be combined, as they take place at successive times in the execute cycle. This forms the "shift data and reset flag" microinstruction, with mnemonics KRB (input) and LDB (output), in which there is a 1 in both bit positions 1 and 2. As before, the advantages of the combined microinstruction are the saving in memory space, and an increase in running speed — two instructions for the price of one!

By this stage you are perhaps beginning to wonder just how the computer does all these things. And that is, so to speak, the 64,000 dollar question. In the limited space available here it is not really possible to answer the question fully; however for the present I will try to give you a basic idea of the principles involved. Hopefully this will be sufficient to provide the background necessary for the more detailed analysis which will be developed as we look at each section.

The basic organisation of EDUC-8 is shown in Fig. 6. This is essentially a combination of Figs. 2, 3 and 4, with the addition of a little more detail.

You will hopefully be able to recognise the registers, such as the PC, the AC, the MA, the MB and the IR, together with such sections as the memory, the input-output interface, and the timing pulse generator. The arithmetic and logic section has been broken down into its component parts, which are now identified as a serial adder, a carry flip-flop, and an AND gate.

The various signal paths between the registers and sections are now shown in more specific form, with controlling AND gates indicated in various strategic positions. These are numbered so that we can refer to each conveniently. Note also

that there are four main signal "highways" or bus lines, arbitrarily labelled the A-bus, the B-bus, the C-bus and the D-bus. The main control signals feed to each register and section have also been indicated.

Essentially, the computer functions by means of control signals applied to the registers and the gates controlling the various signal pathways. And the control signals are formed by suitable combinations of signals from three sources: the timing pulse generator, the major state generator, and the instruction decoder.

At first sight this seems a delightfully general and vague statement, I know. The only real way to give it more meaning is to look at a specific example. Hopefully you will then begin to see the general principles involved, and will be able to visualise more of the machine's operation for yourself.

Let us consider the fetch cycle, in which as we have seen the computer must perform the operations necessary to read out the next instruction from its memory location, and decode it ready for execution. You may recall that the address of this instruction is contained by the PC register, prior to the start of the fetch cycle.

What happens is this. At the start of the fetch cycle, the carry flip-flop is set to contain a 1 by means of a signal applied to its direct set input. This signal is produced by combining the "fetch" signal from the major state generator with a suitable timing pulse from the timing pulse generator.

Following this gate 1 is opened for five clock periods, by means of a control signal produced by combining the "fetch" signal with another signal from the timing pulse generator. And because gate 1 connects the output of bit 0 of the PC register to the C-bus, to which the bit-4 input of the MA register is connected, this has the effect of connecting the PC output to the MA input.

The same control signal used to open gate 1 is also used to admit clock pulses to the PC and MA registers, so that they are simultaneously activated as shift registers. Thus the number in the PC is shifted into the MA, ready to be used to read out the instruction.

At the same time, the same control signal is used to open gates 2 and 3, connecting the C-bus to the A-bus through the serial adder. This has the effect of connecting the output of the PC back to its input, via the adder. And the control is used also to admit clock pulses to the carry flip-flop, so that it can take part in the operation.

The effect of this second set of operations is that the number initially in the PC is not only shifted into the MA, but is also shifted back into the PC — incremented. The incrementing takes place because the carry flip-flop was set to contain a 1, before the shifting took place. Hence the number in the PC is now incremented, ready to provide the address of the next instruction — needed for the next fetch cycle.

During the next part of the fetch cycle, another control signal of eight clock periods duration is produced by combining the "fetch" signal with a suitable signal from the timing pulse generator. This control signal is then used to enable the memory, to open gate 4, and to admit clock pulses to the MB register. The effect of all this is to cause the instruction number in the memory location specified by the MA to be read out of the memory, bit by bit, and shifted into the MB register.

Following this yet another control signal is produced, lasting for a single clock period,

and again formed by combining the "fetch" signal with a suitable signal from the timing pulse generator. This control signal is used to enable the IR register and decoder (the two are actually combined in a single IC), so that the operation code formed by bits 7, 6 and 5 of the instruction are taken from the MB, stored and decoded.

Finally, the last operation of the fetch cycle takes place. This involves the production of yet another control signal, formed by combining the "fetch" signal as before with both a suitable signal from the timing pulse generator, and a signal from the instruction decoder which indicates whether or not the instruction fetched is a memory reference type or not. The control signal thus formed is used to open a gate (not shown) which connects the output of bit 4 of the MB register to a flip-flop in the major state generator known as the "defer status" flip-flop.

The purpose of this last operation is to signal the presence of an indirect addressing memory reference instruction, if one is present, to the major state generator. This is so that the major state generator will enter a defer cycle, if necessary, before starting the execute cycle.

I hope this example will start to give you an idea of what was meant by the bold face paragraph above. Although we won't be able to trace through the operation of the machine during the defer cycle or the execute cycle — particularly for each of the various types of instruction (!) — you may by now be able to see the basic principles involved.

Table 1 gives an analysis of the steps involved in each of the various machine cycles. Using this and the example just given as a guide, you should hopefully be able to trace out what happens in each case.

In a nutshell, the various major states or cycles are defined by the signals generated by the major state generator. Each cycle is then effectively split into a number of segments by the timing pulse generator, which produces appropriate signals lasting various numbers of clock periods. The signals from these two sections are then used, together with signals from the instruction decoder, where appropriate, to produce control signals which are applied to the various signal gates and registers. That's all there is to it!

As the table shows, each cycle lasts for a total of 24 clock pulse periods. Not all of these periods are used to generate timing signals and initiate operations, but the 24-period duration of the cycles has been used to simplify the circuitry of the timing pulse generator. At the two alternate clock pulse rates provided, each cycle lasts for 48 microseconds or 12 seconds, respectively. Thus at the fast rate, a normal fetch-execute sequence lasts for 96 microseconds and a fetch-defer-execute sequence 144 microseconds.

You will perhaps have noticed from the table that besides the fetch, defer and execute cycles, there are two further cycles which we have not yet mentioned. These are marked **deposit** and **examine**, and perhaps their names have already suggested their purpose.

Basically, these are both special "one-shot" cycles mainly associated with control switches on the front panel of the machine. Deposit is used to manually enter numbers (such as instructions) into selected memory locations, while examine is used to manually read out the contents of selected memory locations, for checking. Both are used

TABLE 1 — CYCLES &amp; INSTRUCTION TIMING

CYCLE	T0	T1	T2 —	T5 — T9	T10	T11	T12	T13	T14 — T17 — 21	T22	T23
<b>FETCH</b>	Clear MA, IR, carry (also mem strobe)	set carry		PC to MA, PC + 1 to PC			clear carry		Read instruction into MB	Load IR (last half)	set defer status, MA4 to MB4 if m. ref. instruct.
<b>DEFER</b>	Clear MA, carry, mem strobe	MBO-4 to MA					ditto		Read operand address into MB		
<b>AND</b>	ditto	ditto	Read out operand into MB, AND with AC, result into AC				ditto				
<b>TAD</b>	ditto	ditto	Read out operand into MB, 2's add with AC, result into AC				ditto				
<b>ISZ</b>	ditto	ditto, set carry	Read out operand, increment and put result into MB				ditto	Set carry if MB=0	Write contents of MB, circulate PC through adder		
<b>DCA</b>	ditto	MBO-4 to MA	Write contents of AC, load into MB also				ditto				
<b>JMS</b>	ditto	ditto	Write contents of PC, also load MB				ditto	Set carry	Shift MA into PC via adder		
<b>JMP</b>	ditto	ditto					ditto		Shift MA into PC (via adder)		
<b>SKF, SDF</b>	ditto						ditto	Set carry if flag=1	Circulate PC through adder		
<b>KRS, LDS</b>	ditto		Shift data between device and AC				ditto				
<b>RKF</b>	ditto						ditto	reset i/p flag			
<b>RDF</b>	ditto	reset o/p flag					ditto	reset o/p flag			
<b>IAC</b>	ditto	set carry	Circulate AC through adder				ditto				
<b>RAL</b>	ditto	shift AC one bit L					ditto				
<b>CMA</b>	ditto		Circulate AC through inverter				ditto				
<b>CLA</b>	ditto		Shift contents out of AC				ditto				
<b>HLT</b>	ditto						ditto				reset run flag
<b>RAR</b>	ditto						ditto	Rotate AC one bit R			
<b>SMA</b>	ditto						ditto	Set carry if AC = -	Circulate PC through adder		
<b>SZA</b>	ditto						ditto	Set carry if AC = 0	Circulate PC through adder		
<b>DEPOSIT</b>	ditto	set carry		PC to MA, PC + 1 to PC			ditto	Load MB from SR	Write contents of MB, restore in MB		reset run flag
<b>EXAMINE</b>	ditto	ditto		ditto			ditto		Read into MB		reset run flag

primarily for loading in programs via the console switches, and then checking that the instructions have been loaded in correctly.

Associated with the deposit and examine

switches on the front panel of the computer is a third switch, marked **load address**. The function of this switch is to allow a starting memory address to be loaded into the PC register prior to either depositing,

examining, or setting the machine running.

The remaining controls will be discussed next month, when we start dealing with the construction of the machine.

Stay with us!

# Our EDUC-8 computer: starting construction

Having introduced our unique digital computer project and described how it works, the author now starts to describe its construction. Both circuit and wiring details are given for the power supply, front panel board and mother board sections, which together form the foundation of the machine.

by JAMIESON ROWE

EDUC-8 is built into a case of about the same size and shape as a medium-powered stereo amplifier. The case measures 29.3cm wide, 10.3cm high and 35.7cm deep — or 11.5 x 4 x 14 inches, if you prefer. The prototype case is made from 20 gauge steel, and is finished in chocolate hammertone lacquer.

The case is made in two sections, with the main section forming its base, sides and rear. The front and top are provided by the second section, which is designed to slide on from the front. Once assembled the two are held together by a lip at the rear and by four small self-tapping or "PK" screws.

No components whatever are mounted on the front panel-top section of the case, which serves purely as a cover and control escutcheon. This has been done deliberately, so that the machine may be operated easily with the cover removed, to facilitate servicing.

The power supply wiring of the machine is built into the rear of the case, with the two series-pass power transistors and their finned heatsinks mounted on the case rear

itself. Also mounted on the case rear are the sockets for interconnection with the input and output devices.

The actual computer proper is an assembly of printed wiring or "PC" boards, mounted in the front section of the case. Two of the PC boards are mounted vertically, spaced behind the plane of the case front panel by about 2cm and 3cm respectively. The remaining six boards plug into edge-connector sockets mounted on the second of the two vertical boards, to form a vertical stack array.

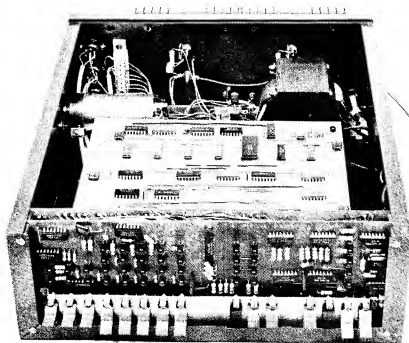
The vertical board nearest the front panel plane performs all of the wiring and interconnections between the front panel components — console switches, LED indicators and so on. For fairly obvious reasons it has been dubbed the "front panel board". The second vertical board makes virtually all of the interconnections between the plug-in PC board sockets, and is known as the "mother board". The copper etching patterns for these two boards are coded EB/F and EB/C respectively; both measure 28.3 by 9.5cm.

In vertical order from the top down the plug-in boards respectively provide the circuitry for the run control, major state generator and timing pulse generator (EB/T); the instruction register and decoder (EB/D); the memory, memory address and memory buffer registers (EB/M); the program counter and serial adder (EB/P); the accumulator (EB/A); and the input-output interfacing circuitry (EB/IOT). Each of these six boards measures 21.5 by 16cm, and the etching pattern codes are as shown in the brackets.

Space has been left in the case so that the plug-in PC boards may be easily removed or replaced. The space also allows any one board to be "extended out" for convenient access during operation, by means of an extender board. This is simply a PC board

## It's growing!

Due to the falling prices for memory ICs, it is now possible to provide EDUC-8 with a 128-word memory at virtually no more than the original cost of the 32-word version. In view of this we are modifying the design, so that as actually described EDUC-8 will have a 128-word memory — four times the original size. Provision is also being made for expansion to 256 words, if desired.



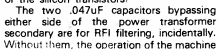
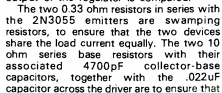
with etched contacts along one end, with conductor strips connecting these to the pins of edge connector sockets mounted at the other end. Thus by plugging the extender board into the computer mother board socket(s) normally occupied by a board, and plugging the board in turn into the extender board sockets, the board is brought out for convenient access while leaving it still connected into circuit. The extender board measures 24.5 by 18cm, and its pattern is coded EB/X.

To remove a board and replace it with the extender board, the boards above it must be temporarily removed to allow access. The desired board is then removed, and replaced by the extender board — with the sockets on the latter facing upward. The other boards can then be replaced, and finally the displaced board plugged into the sockets on the extender board. It then protrudes vertically from the board stack, allowing ready access to both sides.

With the exception of the extender board, which is made from normal SRBP, all of the PC boards in the prototype are made from epoxy fibreglass laminate, and this is probably preferable in view of its greater

*A view of the machine with the top/front panel removed. Note that the front panel board visible is an early version.*

The regulator circuit uses conventional discrete transistors, rather than an IC. This has been done partly because there have been, and still are to a certain extent, supply problems associated with some of the most suitable regulator ICs. A further reason is that I have found it rather difficult to come up with a 6 amp regulator circuit using an IC, which is as stable as this simple discrete circuit, without going to considerably greater complexity and cost.



can be disturbed by mains switching transients and other rubbish which can find its way in via the transformer. The filtering provided by the two simple bypasses has been found quite effective, and no further protection should be necessary providing your machine is housed in a similar earthed metal case.

Despite the simplicity of this power supply circuit, its performance is quite good. Output voltage drops by only 0.12V when the load current is changed from 500mA to 6 amps, which is more than adequate for this sort of application. Similarly the output ripple is well down, being less than 50mV peak to peak even at the nominal full load current of 6A. This means that the supply is easily capable of delivering the nominal 3 amps required for the basic computer, with another 3 amps available for powering the logic in peripherals.

Incidentally this supply would perhaps be worth considering in its own right as a general-purpose heavy duty 5V supply for logic development work. It is fairly rugged, and will even take direct shorts across the output providing they are of reasonably short duration.

As there are only a handful of parts used in the supply apart from the larger components such as the power transformer and reservoir electro, it has been wired up in conventional fashion using a small length of miniature tagstrip. Needless to say, the wiring is not particularly critical, the main thing being to wire up the leads carrying the full load current in fairly heavy wire.

The basic wiring of the tagstrip is shown in the small diagram, to serve as a guide. The rest of the power supply wiring should be easy to work out from the close-up photograph. The two series-pass transistors are mounted on individual 10-cm square finned heatsinks, using the usual silicone grease, mica or plastic insulation shim and sleeved screws to electrically isolate the device cases from the heatsinks and earthed frame.

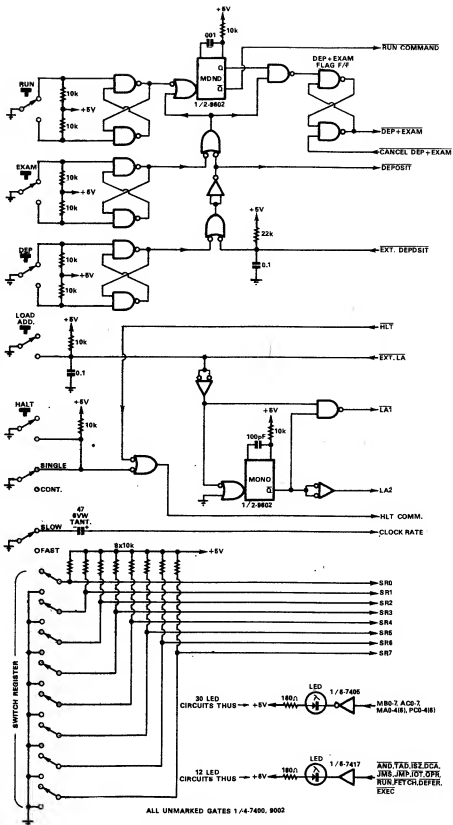
Don't forget that the mains flex should enter the case through a grommetted hole, be clamped upon entry, and have the active and neutral wires terminated in a screw connector strip. The earth wire should be soldered to a lug screwed to the case, to ensure a reliable machine earth.

Note that the PNP driver transistor should have a small clip-on heatsink. This can be one of the fancy moulded types if you like, although a few square centimetres of 18 or 20 gauge sheet brass bent into a suitable "flag" or "9" shape will do just as well.

As soon as the supply is wired up and you have checked to make sure that no obvious errors have been made, it can be switched on — or "powered up", to use the appropriate computerese — and its output voltage checked. Regulation and ripple can also be checked if you have the facilities, although if the open-circuit output is right the rest will probably be in order also.

As with most simple supplies, the open circuit voltage tends to be a little higher than when even a small load current is drawn. It should measure between about 5.3 and 5.4 volts, dropping to about 5.12V at about 500mA drain and then dropping much more slowly to around 5.00V at 6A. If your supply is consistently higher or lower than these figures, the cause will almost certainly be due to the zener tolerance.

The remedy is simple: adjust the value of either the 120 or 470 ohm resistors across the zener, using higher value shunt resistors,



EDUC-8 FRONT PANEL CIRCUITRY

until the desired voltages are produced. Pad down the 120 ohm resistor in this way if the voltage is low, or the 470 ohm resistor if the voltage is high.

With the power supply operational, you will then be in a position to start on the actual computer itself. I suggest that you begin with the front panel board, as this will

let you see how the finished machine is going to look. And with the board complete and hooked up to the power supply, you will have an impressive array of lights and switches to demonstrate your progress to others!

The circuitry which is associated with the front panel board is shown in the diagram,



## EDUC-8 computer

SRO-6 inclusive (or SRO-7 inclusive with the extended memory). The two signals are LA1 (negative logic), which is used to clear the PC, and LA2 which is used to perform the actual loading.

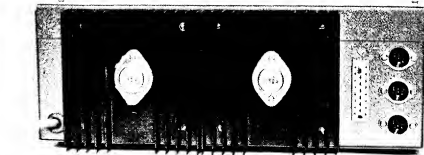
As may be seen, the circuit which produces the two signals is quite simple, using a two-input gate, the second one-shot element from the 9602 device, and a further gate used as an inverting buffer. The one-shot is arranged to trigger only on negative-going edges of the inverted (and therefore positive logic) LA signal, so that if the load address key switch were bounceless the one-shot would not operate until the key were released.

If we ignore bounce for a minute, then, it can be seen that the Q-bar or complementary output of the one-shot will remain in the high logic state while the LA signal is present, because the one-shot will not yet have triggered. As a result the gate producing the LA1 signal will have two high outputs, and will produce the signal to clear the PC register.

Then, when the key is released, or the external LA signal removed, the input of the gate connected to the LA signal will go low, causing the gate output to go high and thus inhibiting the LA1 signal. But at the same time the one-shot will trigger, causing the Q-bar output to go to the low state for a short time. As a result the inverting buffer attached to this output will produce the LA2 pulse signal, causing the address set up on the switch register to be loaded into the freshly cleared PC register.

Although you might think that this neat chain of events would be fouled up by contact bounce in the LOAD ADDRESS key, this isn't so. If you trace through the circuit, you'll see that the only effect of bounce is to cause the sequence to be repeated a number of times in rapid succession. And because the last "bounce" after the key is released will always by definition cause a negative-going transition at the input of the one-shot, the last event will always be the generation of an LA2 pulse. Hence despite bounce, correct loading always takes place.

It is because the load address circuit is unaffected by bounce that the LOAD ADDRESS key is not provided with a bounce suppression flip-flop as fitted to the



The rear of the case, showing the IOT connectors and regulator transistors.

RUN, EXAMINE and DEPOSIT keys. There is no point in having a flip-flop if it is not necessary.

One of the three remaining switches on the front panel board is the SLOW/FAST switch, which in the slow position completes the earth return circuit of a 47µF tantalum electrolytic capacitor. The other end of the capacitor connects to the main clock pulse generator on the timing board, to alter its operating frequency.

The remaining two switches are the HALT key and the SINGLE/CONTINUOUS switch, which as may be seen are both in parallel. Both have the effect of producing a low logic level at one input of a gate acting as an OR element, to produce a logic signal labelled HALT COMMAND. The second input of the OR gate is fed from the HLT instruction output line (negative logic) of the instruction decoder, so that the same HALT COMMAND signal is generated in the event of a "halt" instruction.

The effect of the HALT COMMAND signal is to cause a reset pulse to be fed to the run control flag flip-flop in the timing circuit, at the beginning of the second half of T23 of the next EXECUTE cycle. This in turn causes the main run control flip-flop to the reset at the end of T23, so that the machine stops running after fully completing the current fetch-execute cycle.

The run control flag flip-flop can only be reset during T23 of an execute cycle, and then only if the HALT COMMAND signal is present. This has a number of implications, some of which will be discussed later. One implication is that it is quite in order for the

RUN and HALT keys to be pressed at the same time — contradictory though this may sound. Since the HALT key is effectively only sensed during T23 of the execute cycle, pressing both keys together and holding them both down simply causes the machine to run for a single complete fetch-execute (or fetch-defer-execute) cycle.

In other words, it performs a single instruction step — nothing more, nothing less. So that if the HALT key is held down continuously and the RUN key pressed repeatedly, the machine will simply step through a program one instruction at a time. This can be very handy for analysing the operation of a program — particularly if it is not doing what you expected!

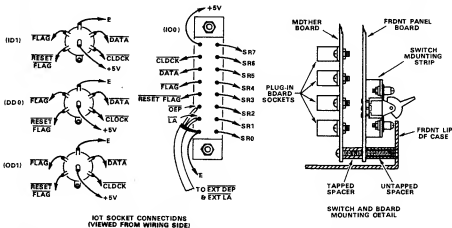
It can be rather tedious having to hold the HALT key down for any length of time, though, and this is why the SINGLE/CONTINUOUS switch is provided. In the single step position, it is simply equivalent to holding the HALT key down all the time — but a little easier on the operator's finger.

Wiring of the front panel board should be found a fairly straightforward task, if you use the wiring diagram as a guide together with the circuit and the photograph. Most of the wiring should be fairly self explanatory, although care should be taken with such matters as correct orientation of the indicator LEDs and the various ICs.

Note that provision is made on the board for indicator LEDs corresponding to bits 0-6 inclusive of the PC and MA registers. This is sufficient for the 128-word memory, where only 7 address bits are used, but will not allow indication of the eighth bits (PC7 and MA7) which become active if the memory is expanded to 256 bits. Indicators for these bits will be described later.

Probably the only other point to mention about the front panel board at this stage concerns the mounting of the switches. To enable these to be replaced individually at some future time, if they become faulty, they have been mounted on a small metal strip, which is attached to the front of the PC board via small screws and nuts, at each end. The switches are fastened to the strip from the front using the screws and nuts supplied with them, and with their bodies passing through clearance slots. Finally their rear lugs pass through holes in the PC board, and are soldered to the appropriate copper pads.

This technique should allow any of the switches to be removed later on if necessary, by undoing its mounting screws



At left are the IOT connector wiring details, and at right the switch and vertical board mounting arrangements.



an unsoldering its lugs using a small instrument iron introduced between the front panel and mother boards. A little tricky, to be sure, but not too difficult. The small diagram shows the arrangement.

There is little to say about the mother board, particularly in terms of its circuit function, because its main function is to perform all of the interconnections between the socket pins of the various plug-in boards.

Apart from this very necessary and worthwhile function, it also brings out the connections between the plug-in boards and the front panel circuitry, and between the IOT board and the input-output device sockets. It also reticulates the 5V supply power, and mounts the four resistors used to terminate the A,B,C and D data bus lines.

Whereas the twin PC edge connectors used for all of the lower boards have their "inner" ends open, to clear the board, the single top board socket has both ends closed.

The wiring of the mother board should again be fairly evident from its wiring diagram. The connections between it and the front panel board are all marked identically, so that when the two boards are wired up and mounted together via six half-inch tapped spacers, it is simply a matter of joining up the pads with the same markings. In many cases these are exactly opposite one another, so that interconnection involves only a short length of tinned copper wire bridging the gap. Where this is not possible, the leads are generally in the same order, but merely a little further along so that a short length of hookup wire will be needed.

The only exceptions to this rule are the connections to the PC and MA indicators, which are separate on the mother board but interleaved on the front panel board. This is a little more tricky, but if extra is wired separately and with care, all should be well. Note that there are no pads on the mother board for the PC and MA register bit 5 and bit 6 indicators — these will be dealt with separately, at a later stage. For the present leave the pads unconnected.

Note that there are four groups of four pads at the bottom of the mother board, which provide the input-output device signal interconnections. These should be connected to the appropriate rear panel sockets, using the small connection cable as a guide. There are three small 6-pin DIN sockets, two of which are used for the output devices while the third is used for input device 1 (ID1).

A larger 16-way socket is used for input device 0 (ID0), and the other pins of the socket are connected to the switch register pads SRO-7, the external deposit pad and the external load address pad — the last two being on the front panel board. This allows the ID0 socket to be used for input devices incorporating a hardware loader, such as a paper tape reader. Note that the external DEPOSIT and LA connections between the socket and the front panel board should be made in twin shield wire, to prevent possible malfunction due to spurious pickup.

In addition to control and data signal connections, each input-output device socket also receives 5V supply power direct from the regulator output. This avoids trouble due to supply bus transients.

The 5V supply connections to the mother and front panel boards should be wired in fairly heavy leads, to ensure low voltage drop. The stranded-conductor leads from a

## EDUC-8 PARTS LIST — 1

### MAIN CASE AND POWER SUPPLY

- 1 Case and lid/front panel, 29.3 x 10.3 x 35.7mm (W x H x D), with switch mounting bracket
- 1 Power transformer, 240V/10V at 6A (Ferguson type PF3798, Jones type JT 139 or similar)
- 2 10cm square finned heatsinks, flat mounting type
- 3 6-way DIN sockets (McMurdo type 1290-06-01)
- 1 16-way polished plug and socket (McMurdo type 1338-12-02, 1338-02-02)
- 1 Silicon rectifier bridge, PB40 or similar
- 2 2N3055 NPN power transistors with mounting accessories
- 1 TT800, BFS92, AY9139 or similar medium power PNP silicon
- 2 BC108, BC208, BC548 or similar general purpose NPN silicon
- 1 82K9/10 CV2 or similar 6.2V 400mW zener diode
- 2 4700pF LV polyester or polycarbonate
- 1 .022uF LV polyester, etc
- 2 .047uF LV polyester, etc
- 1 470uF 10VW electrolytic
- 1 15,000uF or 22,000uF 16VW electrolytic with mounting clamp
- Resistors: 2 x 0.33ohm 5W, 2 x 10ohm 1W, 2 x 100ohm ½W, 1 x 120ohm ½W, 2 x 220ohm ½W, 1 x 470ohm ½W
- Mains cord and plug; grommet and plug for same; 2-way "B-B" connector strip; 7-lug section of miniature resistor panel; 2 x 2-lug miniature tagstrips; 2 x 12.7mm tapped spacers; rubber feet for case; screws, nuts, washers, solder lugs, etc.

### FRONT PANEL BOARD

- 1 Printed wiring board, code E8/F (28.3 x 9.6cm)
- 8 SPDT paddle switch, red paddle (C&K type 7101)

- 2 SPDT paddle switch, grey paddle (C&K type 7101)
- 3 SPDT spring return paddle switch, red paddle (C&K type 7108)
- 2 SPDT spring return paddle switch, black paddle (C&K type 7108)
- 4 spacers, untapped, 17mm long
- 4 7400 or 9002 quad gate
- 5 7405 or 9017 hex inverter (open collector)
- 2 7417 or 9N17 hex buffer (open collector)
- 1 9602 dual one-shot
- 42 Low cost red LED's single ended type (CLD419, FLV110, 5082-4850, 5082-4484, SL103, CQV24 or similar)
- 1 100pF NP0 ceramic capacitor
- 1 .001uF LV polyester or polycarbonate
- 6 0.1uF LV polyester or polycarbonate
- 1 47uF 6VW tantalum electrolytic
- Resistors: 42 x 180ohm 1/4W, 1 x 1k 1/4W, 18 x 10k 1/4W, 1 x 22k 1/4W
- Mounting screws for switch bracket; hookup wire for links; solder, etc.

### MOTHER BOARD

- 1 Printed wiring board, code E8/C (28.3 x 9.6cm)
- 4 32-way edge connector sockets, gold PC tail clips (McMurdo type 133-14-17) One socket closed both ends (closed end foot type 4862-01-08)
- 7 16-way edge connector sockets, gold PC tail clips (McMurdo type 133-12-17)
- 3 0.1uF LV polyester or polycarbonate capacitors
- 2 6800hm 1/4W resistors, 2 x 4700 ohm 1/4W resistors
- Hookup wire for links, connections to front panel board IOT connectors and power supply; 6 x 12.7mm tapped spacers for assembling boards; screws, etc.

length of 3-core plastic insulated mains flex are quite suitable.

The mother board-front panel board assembly is mounted into the main case by four ¼in Whitworth by 1 inch countersink-head screws, two at each end in the corners. The screws pass through countersunk holes in the wide turnover lips at the front of the case, and are then fitted with clearance spacers approximately ¼ inch long. When the board assembly is in position, the screws then pass through the holes in the front panel board, and mate with the tapped holes in the spacers between the two boards. Tightening the screws carefully thus completes the assembly of the two boards, and also fixes them rigidly behind the final plane of the front panel.

With these boards wired up, connected to the power supply and mounted in the case, you can power up again and check progress. It won't be easy to check the operation of the control switches at this stage, but you can certainly check the LED indicator circuits and the operation of the switch register switches.

When you switch on, all of the register indicator LEDs on the left-hand side of the front panel board should light, while the rest should all remain dark. Then, with a piece of hookup wire with one end connected to power supply negative (ie, the case), it should be possible to turn off each of the

register LEDs by touching the appropriate interconnecting link with the wire.

With the same wire it should be possible to light up any of the normally dark LEDs, by touching their interconnection links or pads.

If any of the register LEDs don't come on when power is applied, look for a wiring error. The most likely cause will be a LED wired in the wrong way around. Failing a wiring error, you could have a faulty LED or IC, which will have to be replaced.

Similarly if any of the instruction indicator or machine state LEDs on the right-hand side of the board can't be turned on by entering the input of their buffer, again look for a wiring error or a faulty component and remedy it as soon as possible.

Finally, you can check the operation of the switch register by temporarily connecting its inter connection pads to the input pads of one or more of the register indicator drivers. You can do this either one switch at a time, or all at once — say by connecting them to all of the MB or AC register drivers. In the up position each switch should cause its indicator LED to light, while in the down position the LED should be extinguished.

Barring a faulty switch or pullup resistor, all should be well at this stage. You should now be ready to tackle the timing and control board, which will be the next section described.

# Building our computer: three more sections

The description of our unique digital computer project continues here with construction details for the timing and run control board, the instruction decoder board and the accumulator board. The author also explains how to check the operation of these boards, in preparation for the addition of the remaining sections.

by JAMIESON ROWE

The timing and run control board is the uppermost of the EDUC-8 plug-in board stack, and is coded BB/T. The circuitry on this board comprises the master clock oscillator, the run control logic, the timing pulse generator, and the major state generator. The logic diagram for these sections of the machine is shown in Fig 1.

The master clock oscillator uses a circuit which I have used in previous digital projects, and found to be stable and reliable. It is basically a relaxation oscillator using an RC integrator feedback circuit around a Schmitt trigger formed from half a 7413 device. A general purpose NPN silicon transistor (BC108, BC208, BC548, etc) is used as an emitter follower to reduce loading on the integrator. This makes it possible to generate stable frequencies as low as 0.1Hz using practical values of capacitance.

In this case the basic oscillator uses a 10k resistor and 220pF capacitor, giving a "fast" clock frequency of approximately 500kHz. To produce the "slow" clock frequency of 2Hz, the appropriate front panel switch connects a 47uF tantalum electrolytic capacitor in parallel with the 220pF. The connection to the switch and capacitor is made via contact pad 7 of the board edge connector, as shown.

The second half of the 7413 device is used as a buffer between the master clock oscillator and the run control logic. At the heart of this latter section is the run control flip-flop; this is a J-K element — half of a 7473 device — which controls the main gate admitting the clock pulses to the timing circuitry.

The clock input of the run control flip-flop is driven by the master clock pulses themselves, so that it operates the main gate synchronously. This ensures that the gate always opens and closes to pass complete clock pulses, and never fractional pulses. Opening and closing of the main gate is performed by the run control flip-flop in response to control signals applied to its J and K inputs by a second flip-flop, designated the "run flag".

The run flag flip-flop is a simple R-S type formed from two 7400 gate elements. It is initially preset to the state which produces a low logic level at the J input of the run control flip-flop, so that the main gate is closed. However a negative logic RUN COMMAND pulse fed to the run flag from the front panel circuit via edge connector pin 4 will cause the flag to switch to the set state, applying a logic high to the J input of the run control flip-flop. At the completion

of the next master clock pulse the latter will therefore change state, opening the main gate to begin the machine running.

Before we go any further, let me digress briefly for a moment to introduce some abbreviations which I will be using frequently in the remainder of the present description, and those to follow. Without these abbreviations, the description would become both unwieldy and difficult to follow.

No doubt you will have noticed already that some logic signals we have encountered use the positive logic convention (high equals true), while others use the negative convention (low equals true). This occurs throughout the machine, the logic conventions for each signal having been determined in order to simplify the logic and minimise the number of devices. In some cases the same signal is used in both positive and negative logic forms, for different purposes.

To simplify discussion from now on, a negative logic convention on the particular signal or signal version being referred to will be shown by means of the letter L in brackets. Lack of this symbol will therefore imply the positive logic convention. Hence RUN and RUN (L) would be the positive logic and negative logic versions of the same signal, respectively.

Other abbreviations which will be used are "FF" to stand for flip-flop, and "MCP" for master clock pulses. There will be others too, but these will be introduced as we go along. Note that negative logic signals are shown on the diagrams by a bar over the designation, and also by a "bubble" at logic element inputs and outputs. The diagrams also show the logical OR operation as a plus sign, whereas this will be spelled out in the text as the symbol is not available in our typesetting.

We should now be able to continue, in a slightly more elegant manner.

Halting of the machine is achieved by applying a negative pulse to the second input of the run flag FF, to change its state and apply a logic high to the K input of the run control FF so that it will close the main gate at the end of the current MCP.

Two different signals are used to reset the run flag FF for halting, the two being applied via a pair of 7400 gate elements connected to form a negative logic OR gate. One signal is a turn-on reset signal shown on the diagram as R(L), which will be discussed further shortly. The second signal is the normal "halt" signal (L), used to stop the machine running at the end of a deposit or

examine cycle, or at the end of an execute cycle in response to a front panel control or a "HALT" instruction.

The halt (L) signal is produced by a 2-input gate, one input of which is fed by a positive logic pulse produced by the timing generator during the second half of T23, and accordingly shown as T23.5. The second input of the gate is fed from another gate which performs an OR between the DEP OR EXAM (L) signal from the flag on the front panel board, and the output of a further gate which ANDs the HLT COM signal from the front panel board with an EXECUTE signal produced by the major state generator.

This sounds more complicated than it really is, as the diagram shows. All it means is that during normal running, the run flag FF is turned off at time T23.5 if the machine is either performing a deposit or an examine, or if it is performing an execute cycle and the HLT COM signal is present.

Both the run control and run flag FFs are reset initially when power is applied to the machine, to prevent it from running until this is specifically commanded. This turn-on reset function is performed by the R(L) signal, generated by a simple circuit using a 7400 gate element whose two inputs are taken to earth via a parallel combination of a 100k resistor and a 47uF tantalum electrolytic capacitor.

When power is first applied, the capacitor is an effective short-circuit to ground applied at the gate inputs. This produces a logic high at the gate output, and hence the R(L) signal at the output of its following buffer inverter (X7420). However the capacitor soon charges through the gate input circuit, allowing the gate inputs to rise to the high level. The R(L) signal therefore disappears after it has served its purpose, and does not appear until the next time power is first applied after the machine has been off.

The purpose of the 100k resistor is to act as a bleed, so that the charge on the capacitor leaks away reasonably soon after the machine is turned off. At the same time, the RC time constant has been chosen so that the turn-on reset signal is not generated unless power to the machine has been interrupted for at least 2 or 3 seconds. This means that very short mains interruptions which would not upset operation, due to the reservoir action of the power supply electro, do not cause a reset to be generated.

As well as being used to reset the run flag and run control FFs upon turn-on, the R(L) signal is also used to perform the same function for the main timing counter and the other FFs in the timing and major state generator logic.

The main timing counter is a Fairchild 9316 4-bit synchronous binary counter, which receives master clock pulses directly from the main gate. A synchronous counter is necessary here, to ensure that all four counter outputs change state



## EDUC-8 computer

simultaneously. This allows groups of counter states to be combined to form continuous timing signals lasting for a number of MCP periods. If a normal non-synchronous counter is used, the small delays between outputs causes "notches" in the resultant timing signals, which can upset machine operation.

The actual timing signals are generated from the various states of the timing counter by means of a 9311/74154 decoder with a network of gates connected to the first 12 of its 16 outputs. The twelfth or 11(L) output of the decoder is also fed back to the parallel enable or PE(L) input of the 9316, effectively causing it to be reset after every 12 clock pulses. This establishes the 12-pulse sequence on which the various cycles of the machine are based.

The cycles are actually 24 clock pulses long, so that for each machine cycle the timing counter and decoder run twice through their basic 12-pulse sequence. Identification of the "first half" and "second half" sequences of the cycles is performed by the sequence counter FF, whose clock pulse input is fed with an inverted version of the 11(L) decoder output.

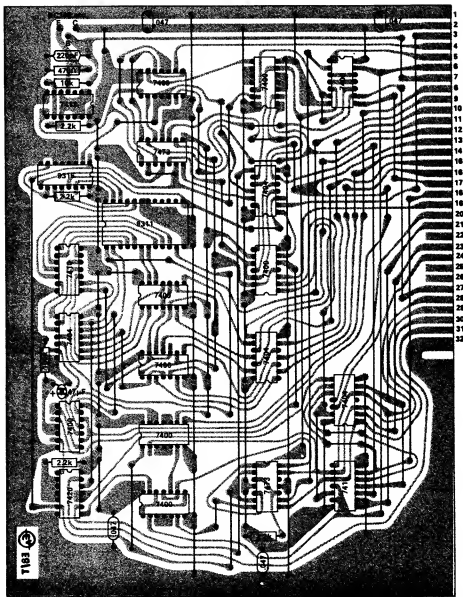
As may be seen, a number of gates connected to the outputs of the timing decoder are used to produce raw timing signals corresponding to various time intervals in the basic timing counter sequence. In most cases the two outputs of the sequence counter FF are then used to separate the raw timing signals into the final timing signals required. The exception is the T0-OR-T12 (L) signal, which because it occurs in both sequences of the cycle is derived directly from the O(L) decoder output.

To illustrate the generation of the other timing signals, look at the logic involved in generating the T2-9 and T14-21 signals. These are both used for serial shifting of instruction end date words, and both last for 8 MCP periods which by design occupy corresponding positions in the first and second 12-pulse sequences of each cycle. Note that the raw signal is first derived by the gates connected to the decoder outputs 2 - 9 (L), inclusive, and this signal is then gated by the sequence counter FF outputs to produce the final T2-9 and T14-21 timing signals.

The signals generated directly by the timing circuitry for use throughout the rest of the machine comprise T0-OR-T12 (L), T0.5 (L), T1, T2-9, T14-21, and T22.5. A further signal T23 is also generated, but this is used solely by the major state generator. Note that there are actually two T2-9 signals and two T14-21 signals, with duplicated logic. This is a carryover from the original 32-word memory version of the machine, where the second sets of gating were required to produce T5-9 and T17-21 signals. Rather than redesign the board completely, the gates have simply been swung over to operate in tandem with the original T2-9 and T14-21 logic.

The major states of the machine are defined by the logic circuitry associated with the two remaining J-K flip-flops on the timing board, those labelled "execute control" and "defer control". This circuitry operates as follows.

When power is first applied, both the execute control and defer control FFs are reset by the R (L) signal. The deposit or



examine flag FF on the front panel board is also reset, by means of a signal produced by the gate whose output connects to pad 17 of the edge connector. This preliminary condition is equivalent to the FETCH state, and accordingly a low level logic signal is produced by a 3-input gate at the FETCH (L) output of the circuit - pad 28. A high level version of the same signal is produced by an inverter and fed to pad 21.

If the machine is set running by means of the RUN key, the first cycle will therefore automatically be a fetch, as required. The next cycle entered depends upon the type of instruction fetched and decoded, and on whether or not bit 4 of the instruction is a 1, in the case of memory reference instructions.

The decision is effectively made by the 4-input gate whose output is connected to the R-S flip-flop marked "defer flag". The inputs to this gate are arranged so that the only situation in which it delivers a negative logic pulse to set the defer flag FF is when a memory reference instruction has been fetched (i.e., not IOT or OPR), bit 4 of the MB register is a 1, and it is the first half of T23 of a fetch cycle. If any of these conditions are not satisfied, the defer flag FF is not set.

In other words, the defer flag FF can only

be set during the first half of T23 of the fetch cycle, and then only if there is both a memory reference instruction fetched, and it is also an indirect address instruction having a 1 in the bit 4 position.

Note in passing that some of the conditions necessary before the defer flag FF can be set are actually tested by the 3-input gate with its output connected to edge connector pad 18. This gate produces a signal designated F.T23.not(OPR OR IOT) (L), which is fed via an inverter to one input of the 4-input gate as well as being fed out for use elsewhere in the machine.

The T23.5 pulse generated by the timing logic is fed to the clock inputs of both the execute control and defer control FFs, so that both FFs are potentially capable of switching to the set state at the end of T23 (which is also the very end of the fetch cycle). However the outputs of the defer flag FF are connected to the J inputs of the two control FFs so that only one can in fact be set. If the defer flag FF remains in its initial reset state, the execute FF will be set; alternatively if the defer flag FF has been set, then the defer FF will be set instead.

Hence the machine is automatically led into an execute or defer cycle, depending upon whichever is appropriate. In either case the FETCH and FETCH (L) signal

outputs are reversed in polarity, to signify that the fetch cycle has ended, and either the EXEC and EXEC (L) outputs are taken high and low respectively, to indicate an execute cycle, or the DEFER (L) output is taken low to indicate a defer cycle.

If the execute control FF is set, the machine enters an execute cycle and performs the instruction concerned. Then at the end of the cycle, the T23.5 timing pulse will reset the execute control FF so that the machine will return to a new fetch cycle.

On the other hand if the defer control FF is set, the machine enters a defer cycle and reads out of the memory the actual address of the instruction operand. At the start of the cycle the defer flag FF is also reset, as the DEFER (L) signal is fed back to the gate and inverter connected to its reset input. As a result, when the end of the defer cycle is reached, the T23.5 timing pulse not only causes the defer control FF to be reset, but also sets the execute control FF. Hence the machine ends the defer cycle and correctly enters an execute cycle.

If the machine is set running normally by means of the RUN key, it therefore continues to run through alternate fetch and execute cycles, with defer cycles automatically interposed between fetch and execute wherever necessary for indirect memory reference instructions. This operation only stops if a halt command is encountered, as a result of the operator pressing the HALT key, or by execution of a halt instruction.

The machine also starts running if either the deposit or examine keys are pressed, as explained earlier. However in this case the deposit or examine flag FF on the front panel will be set, so that the DEP OR EXM (L) signal will be fed to the timing and control board via edge connector pad 6.

This signal has a number of effects. One is that it ensures that the run flag FF is reset by the first T23.5 timing pulse generated, so that the machine automatically halts after a single cycle. At the same time, the 3-input gate which produces the FETCH (L) signal is blocked, so that the machine does not confuse a deposit or examine cycle with fetch. This also has the effect of preventing the defer flag FF from being set during the first half of T23, by blocking the 4-input gate (via the FT23.not(OPR or IOT) (L) 3-input gate).

In addition, the DEP OR EXM (L) signal also prevents the execute control FF from setting at the end of the cycle, by holding down one input of the 2-input gate attached to the latter's J input.

These actions all ensure that if deposit or examine, the machine runs only for a single 24-pulse cycle and then stops without upsetting the major state circuitry. Note that at the end of the cycle, the deposit or examine flag FF on the front panel is automatically reset by means of the CANCEL DEP OR EXM (L) signal produced by the gate attached to edge connector pad 17.

Two further logic and timing signals are derived on this board. One is (P OR DEP OR EXM) (L), fed to pad 24. The other is a version of the same thing gated by T2-9, and fed to pad 19.

Hopefully the foregoing description will have given you a reasonably good idea of the operation of the run control, timing and major state generator logic. It has not been possible to describe the exact function of every gate and inverter, but if this were attempted it would take far more space than

is available — and probably be rather confusing.

The wiring of this board should be fairly self-evident from the diagram of Fig. 2. There are 21 ICs, a few small components and a number of wire links. The main things to watch are the position and orientation of the ICs, and that the links are fitted correctly. It is a good idea to use insulated wire for the links, to prevent accidental shorts. I used single-conductor PVC covered hookup wire.

Don't forget to cut the slot next to pad 32 of the edge connector strip, so that the board can be plugged into the top socket. It may be necessary to file the side of the slot nearest pad 32, and also perhaps the end of the board next to pad 1, to ensure that the board will fit into the socket with all of the pads mating correctly with the socket clips.

The same sort of preparation will probably be necessary with the other plug-in boards, as there will inevitably be small errors in the boards as they come from the manufacturer.

## Decoder board

We can now turn our attention to the second plug-in board, which is that for the instruction register and decoder, with the code EB/D. The logic for this "decoder board" is shown in Fig. 3.

As you can see, the heart of this board is a Fairchild 9334 IC, which actually functions as both the instruction register (IR) and the instruction decoder. This is because the 9334 is what the maker describes as an 8-bit addressable latch. In effect it comprises a 3-bit binary decoder with inputs A0, A1 and A2, whose eight outputs are coupled internally to eight R-S latch flip-flops.

When a low logic level is applied to the C input, all of the internal flip-flops are reset. Then if a low logic level is applied to the E input, whatever logic level is present at the D input will be stored in whichever of the eight FFs corresponds to the decoded 3-bit number applied at the A0, A1 and A2 inputs. In effect, it is an eight-bit memory, where D is the data input, E the write enable input and the three A connections the address inputs.

In this application we clear all the internal

FFs of the device at time TQ.5 of the fetch cycle, by means of the simple gating shown connected to the C input. Then, at time T22.5 in the fetch cycle, after the instruction has been fetched from the memory and has settled in the MB register, a low logic pulse is applied to the E input. As the D input of the device is connected to logic-high, and the three A inputs are connected to bits 7, 6 and 5 of the MB register via edge connector pads R, Q and P, this has the effect that the operation code of the instruction is decoded, and the corresponding 9334 output FF set.

Hence if the operation code is 010 (octal 2), output Q2 would be set. If it is 110 (octal 6), output Q6 would be set. Whatever the operation code, one and only one of the outputs will go high, and will remain high until TQ.5 of the next fetch cycle.

Although the 9334 output concerned thus provides the decoded form of the instruction operation code from T22.5 of its own fetch cycle until TQ.5 of the following fetch, this signal is in most cases only used during the execute cycle. Hence the 9334 outputs are not used directly, but are gated by the EXEC signal from the major state generator. This produces the eight primary instruction outputs, all of which are fed to edge connector pads A-H inclusive, in negative logic form: AND (L), TAD (L), ISZ (L), DCA (L), JMS (L), JMP (L), IOT (L) and OPR (L).

An (OPR or IOT) signal un gated by the EXEC signal is also produced, and fed to connector pad 16. The reason why the signal is not gated by the EXEC signal is that it is used primarily on the timing and control board (pad 20), to prevent the defer flag FF being set for IOT or OPR instructions. To be effective, it must therefore be present during T23 of the fetch cycle.

The additional gating attached to the O6 and Q7 outputs of the 9334 is involved in decoding the augmented operation code of the IOT and OPR instruction formats. Thus the IOT signal is further gated by the signals from MB register bits 0,1 and 2 to produce the CLEAR IOT FLAG (L), IOT SHIFT (L) and SKP ON IOT FLAG signals respectively, fed to connector pads 6, 11 and 7.

Similarly the OPR signal is gated by both

## EDUC-8 PARTS LIST — 2

### TIMING AND CONTROL BOARD

- 1 PC board, code EB/T, 21.5 x 16cm
- 1 BC108, BC208, BC548 or similar NPN transistor
- 9 7400 or 9002 quad 2-input gate IC
- 4 7404 or 9016 hex inverter IC
- 2 7410 or 9003 triple 3-input gate IC
- 1 7413 or 9N13 dual Schmitt IC
- 1 7420 or 9004 dual 4-input gate IC
- 2 7473 or 9N73 dual J-K flipflop IC
- 1 9311 or 74154 16-way decoder IC
- 1 9316 synchronous 4-bit counter IC
- 1 470 ohm ¼W resistor
- 2 2.2k ¼W resistors
- 1 10k ¼W resistor
- 1 100k ¼W resistor
- 1 220pF polystyrene or NPO ceramic
- 4 .047uF LV polyester or ceramic
- 1 47uF 6VW tantalum electrolytic
- Insulated hookup wire for links

### DECODER BOARD

- 1 PC board, code EB/D, 21.5 x 16cm

- 8 7400 or 9002 quad 2-input gate IC
- 2 7404 or 9016 hex inverter IC
- 2 7410 or 9003 triple 3-input gate IC
- 1 7420 or 9004 dual 4-input gate IC
- 1 9334 eight bit addressable latch IC
- 2 2.2k ¼W resistors
- 3 .047uF LV polyester or ceramic
- Insulated hookup wire for links

### ACCUMULATOR BOARD

- 1 PC board, code EB/A, 21.5 x 16cm
- 2 7400 or 9002 quad 2-input gate IC
- 2 7401 or 9012 quad 2-input gate with open collectors
- 1 7404 or 9016 hex inverter IC
- 2 7405 or 9017 hex inverter with open collectors
- 1 7410 or 9003 triple 3-input gate IC
- 2 7495 or 9395 four-bit shift register
- 1 820 ohm ¼W resistor
- 3 2.2k ¼W resistors
- 4 .047uF LV polyester or ceramic
- Insulated hookup wire for links

## EDUC-8 computer

the MB bit 4 and its complement, and in each case then further gated by MB bits 0,1,2 and 3 to produce the eight OPR microinstruction signals CLA (L), CMA (L), RAL (L), IAC (L), SZA (L), SMA (L), RAR (L), and HLT (L). These are fed to connector pads 31 — 24 inclusive, as shown.

The remaining logic circuitry on the instruction decoder board is involved in producing secondary gating and timing signals, by combining the foregoing primary instruction and microinstruction gating signals with timing signals. Thus a T14-21. (JMS OR JMP) (L) signal is produced by combining the JMS (L) and JMP (L) signals in a gate performing the OR function, and then gating this with the T14-21 timing signal. The resultant is fed to connector pad 17, as shown.

Similarly a T2-9. (TAD OR IAC OR CMA) (L) signal is produced and fed to pad 13, and a T2-9 ISZ (L) signal produced and fed to pad 14. A (JMS OR DCA) signal is also produced and fed to pad 15.

Signals corresponding to those instructions and microinstructions which involve PC register incrementing or "skipping" — ISZ (L), SZA (L), SMA (L) and SKP ON IOT FLAG (L) — are also combined by a 4-input gate performing the OR function, and gated with the T14-21 signal to produce a signal designated T14-21.SKP (L), which is fed to pad 18.

And finally, signals corresponding to both the major states and instructions which involve the memory are assembled by a 4-input gate and a 3-input gate, both performing the OR function, and gated with the appropriate T2-9 and T14-21 timing signals. The outputs of the two gates are then combined by a further gate performing the OR function, to produce a signal designated MEMORY ENABLE. This is fed to pad 23.

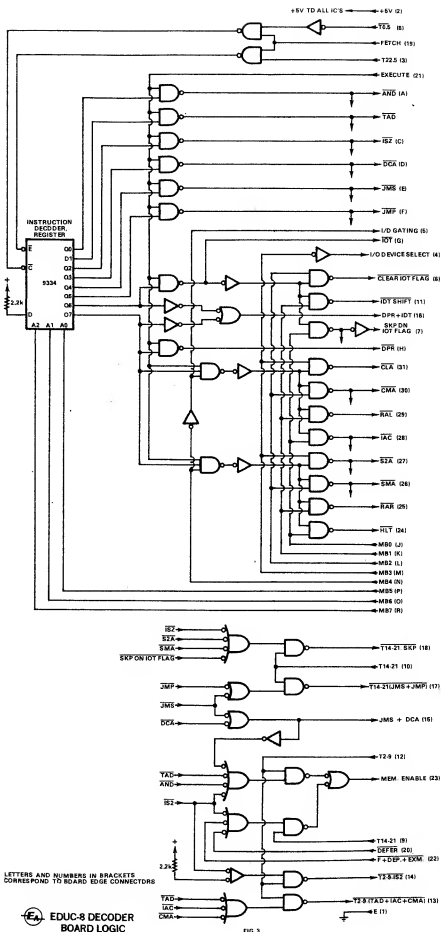
Wiring of the decoder board should be fairly self-evident from the diagram shown in Fig. 4. There are only 14 ICs, five minor parts and again a number of wire links. As before the main things to watch are that the ICs are correctly orientated, in their correct positions, and that the wire links are in their correct positions.

The decoder board plugs into the mother board position second from the top, immediately below the timing and control board.

### Accumulator board

The third section of EDUC-8 to be described at this stage is that comprising the accumulator (AC) register and its associated logic. This section is on the plug-in board coded E8/A, and for convenience described as the accumulator board. The logic diagram is shown in Fig. 5.

As may be seen, the heart of this section is the AC register itself, which is simply formed by two 7495 four-bit shift register ICs connected together to form an 8-bit register. The 7495 devices are internally connected for right shifting, which is of course used for shifting data into and out of the register, and also for the RAR microinstruction. To achieve the left shifting required for the RAL microinstruction, the parallel inputs PA—PD of both devices are connected to the outputs of the "next right" positions. The PD input of the right 7495 is connected to the A output of the left, to



### EDUC-8 computer

complete the loop so that all 8 bits are retained.

The RAL (L) signal input at pad F is taken to the "mode" (M) inputs of the two 7495s, via an inverter to give the correct logic polarity. Thus when the RAL microinstruction occurs, the devices are switched to the parallel loading mode. As the parallel load clock inputs (CP2) are connected to the T1 timing signal input on pad 13, a left shift of one bit thus occurs at time T1 of an RAL execute cycle.

The outputs of all eight AC bits are taken to connector pads J-R inclusive, for connection to the LED indicator drivers on the front panel board. AC bit 0 is also connected to pad 11, for shifting of data to the IOT interface board. Eight open-circuit collector inverters with their inputs connected to AC bits 0-7 and their outputs commoned and taken to the positive rail via an 820 ohm resistor are used to perform the NOR function, so that their output at pad 15 is at logic high level only when all eight AC bits are zero. This signal is used when executing the SZA microinstruction.

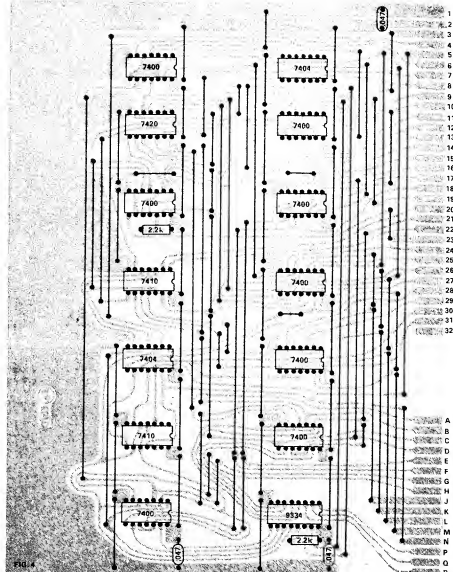
The shift register clock pulse inputs CP1 of the two 7495 ICs are commoned and fed via an inverter from a gate which ANDs master clock pulses arriving at pad 9 with one of a number of signals fed to it via the associated gates. Thus a single clock pulse is fed to the AC at T13 of an RAR microinstruction execute cycle, due to the T13.RAR (L) signal from pad 8. Similarly eight pulses are applied during T2-9 of the execute cycle of a TAD instruction or an IAC CMA microinstruction, due to the T2-9.(TAD or IAC or CMA) (L) signal from pad 14, and so on.

The bit 0 output of the AC is applied to one input of the gate marked AND<sub>2</sub>, the other input of which connects to the D-bus input on pad 7. After passing through an inverter to restore the logic polarity, the output of the AND gate is then gated by an inverted version of the AND (L) signal from pad A. Thus during the execute cycle of an AND instruction, a path is provided for serial ANDing of the number in the AC with the number read from the memory via the D-bus. The resultant is fed to the A-bus, pad 4, and then back to the input of the AC via an inverter.

The remaining logic of this section is involved with connection of the AC bit 0 output to either the B-bus or C-bus (pads 5 and 6), as required, for various instructions and microinstructions. I will not trace through this in detail, to conserve space. However it may be worth noting that the effect of the CLA (L) signal from pad H is to block two of the paths between AC0 and the B-bus, whereas the effect of the other inputs is to enable a path between AC0 and either the B-bus or C-bus.

The logic has also been arranged so that the CLA microinstruction can be combined with either the IAC or CMA microinstructions, and IAC also combined with CMA.

As before the wiring of the accumulator board should be straightforward, using the diagram of Fig. 6 as a guide. There are 10 ICs, some nine minor parts and a number of wire links, and here again the IC orientation and position should be watched carefully to avoid errors. This board plugs into the second bottom position on the mother



board, which is the uppermost of the two positions having two 16-way edge connectors.

When the three boards described in this section have been wired up and checked against the diagrams for errors, they may be plugged into their respective sockets — assuming that all seems well. It will now be possible to test many of the basic functions performed by these sections of the machine, and this is a good idea before you proceed to build up the remaining sections.

## Testing progress

In order to perform the tests it is necessary to make temporary connections between the switch register pads SR0-7 on the front panel board (also available on the 16-way input device connector), and the MB0-7 connector pads on the left-hand end of the mother board. SR0 should be connected to MB0, SR1 to MB1, and so on, down to SR7 and MB7. This is to allow the switch register to be used to "dummy" or substitute for the MB register, as yet unwired.

With these temporary connections made, apply the power. If all is well, the only LED which will light on the right-hand side of the front panel will be that for FETCH, which is

normally lit when the machine is not running. On the left-hand side of the front panel, the LEDs for the PC and MA registers should all be glowing, while those for the MB register should correspond to the positions of the SR switches – up switches producing a glowing LED, and down switches producing a dark LED. The LEDs for the AC register will light in a random pattern at this stage, and have no particular significance apart from reflecting the turn-on bias of the FFs in the AC devices.

For a first test, make sure that you can set the MB register LEDs to any desired binary number by setting it up on the SR switches. This ensures that you have the temporary connections right, so that the SR can indeed be used as a substitute MB register to feed in instructions.

Now set the FAST/SLOW switch to the upper or slow position, and the SINGLE/CONT switch to the upper or single step position. The machine will now be set for slow running, and single stepping. Then set all eight of the SR switches to the down or 0 position, in effect giving the machine an AND instruction directly addressing memory location 0000 (which at this stage does not exist, of course — nor does any other location).

## EDUC-8 computer

You are now ready for the first big test. Watching the LEDs on the right-hand end of the front panel, press the RUN key. The run LED should light, showing that the machine has started running, while the fetch LED should remain lit — showing that it is valiantly trying to perform the fetch cycle, slightly handicapped at this stage by the lack of both the PC register and the complete memory system!

The fetch LED should remain lit for about 12 seconds, then it should go dark and the execute LED should glow, showing that the machine has correctly entered an execute cycle. At the same time the AND instruction LED should light, showing that the correct "instruction" has been decoded and its execution is being attempted.

Both the execute and AND instruction LEDs should remain lit for another 12 seconds or so, whereupon both they and the run LED should go dark, indicating that the machine has stopped running. When this happens the fetch LED should come back on again.

During the execute cycle you may have noticed that if there was a number other than 00000000 in the AC register, it was apparently shifted around eight bits to the

right, to end up where it began. This is normal, as the machine performs the AND operation between the number in the AC and a logic "1" effectively present on the D-bus when the memory board is absent. The resultant is returned to the AC.

If all is well so far, set SR switch 4 to the up position, to simulate the MB4 bit being set to 1 for indirect addressing. Then press the RUN key again. The previous chain of events should now be repeated, except that before entering the execute cycle the machine should spend about 12 seconds with the defer LED glowing.

If this checks out also, set switch SR4 down again and set SR5 to the up position, to simulate a TAD instruction. Pressing the RUN key should produce a repeat of the first sequence, except that this time the TAD instruction LED should glow during execute. The number in the AC will also be lost, as the machine will shift it out to attempt passing it through the serial adder (at present non-existent).

In similar fashion you can check the decoding of the remaining primary instructions, simply by setting SR5, SR6 and SR7 to the appropriate operation code. And you can check that with bit 4 set to a 1 by having SR4 up, the machine will pass through the defer state between fetch and execute, for the six memory reference instructions. You should conversely check that it does NOT enter a defer cycle for the

IOT and OPR instructions, despite SR4 being up!

Finally, you can test those of the OPR microinstructions which change the content of the AC, without requiring the adder. These are CLA, CMA and RAL. The other two content-changing microinstructions (IAC and RAR) can also be partially checked, but at this stage will not work properly because of the absence of the adder.

To perform these checks, first set the SLOW/FAST switch to the down, or fast position. Then set switches SR7, SR6 and SR5 to the up position, to simulate the OPR operation code (octal 7). Leave the SINGLE/CONT switch in the up position, for single step operation.

As the AC register content is probably zero by this stage, the best microinstruction to try first is probably CMA. So set switch SR2 to the up position and switches SR4, SR3, SR1 and SR0 to the down position, to set up the full CMA coding (octal 704). Then press the RUN key, whereupon the AC should be set to 11111111 (equivalent to minus 1 in two's complement binary arithmetic).

Pressing the RUN key again should restore the AC content to all zeroes again, and further presses should simply cause the two AC content situations to alternate back and forth. Make sure that you end the test with the AC content at minus 1, though, so

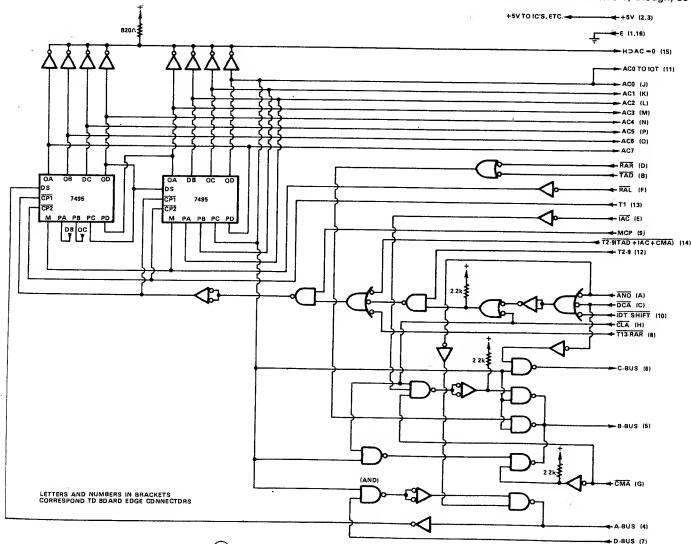


FIG. 5

EDUC-8 ACCUMULATOR BOARD LOGIC



that you are ready to perform the next test.

This time set switch SR2 down, and SR3 up instead. This produces octal code 710, or that for CLA. Pressing the RUN key should now simply wipe out the AC content, leaving it zero. Continued pressing should have no effect.

To test the RAL instruction, it will be necessary to have an AC content other than zero or minus 1, because these both look the same no matter how many times they are shifted left! The easiest way to give the AC a suitable content is probably to turn the power off for about 10 seconds, then turn it back on. The AC content should then come up with a random value, hopefully neither zero nor minus 1.

If you are unlucky, and the AC does come up with zero or minus 1 consistently, there is still a way to produce a suitable AC content. This is by using the RAR microinstruction, which as yet will function only partially but sufficient for our purpose.

First set the SR switches back for CMA (octal 704), and press the RUN key to give the AC a content of minus 1. Then set the SR switches to octal 722, representing RAR (SR7, 6, 5, 4 and 1 up, SR3, 2 and 0 down). Pressing the RUN key a few times should then move zeroes into the AC from the left hand end. Do this until you have about 3 or 4 bits left in the AC set to 1.

Having produced a suitable content in the AC by one or other of these methods, now set the SR switches for the RAL microinstruction (octal 702). Pressing the run key should cause the number in the AC to be shifted one bit position to the left, with the value of AC bit 7 being transferred around into bit 0. Further shifting should be produced by continued pressing of the RUN key.

At this stage you will have tested all of the microinstructions capable of being performed properly by the machine in its incomplete form. You can if you wish test the RAR microinstruction partly, along the lines just described for setting up the AC for the RAL test. You can also try the IAC microinstruction (octal code 701), although at this stage it will simply act in the same way as CLA or DCA — simply clearing the AC of any content and leaving zero.

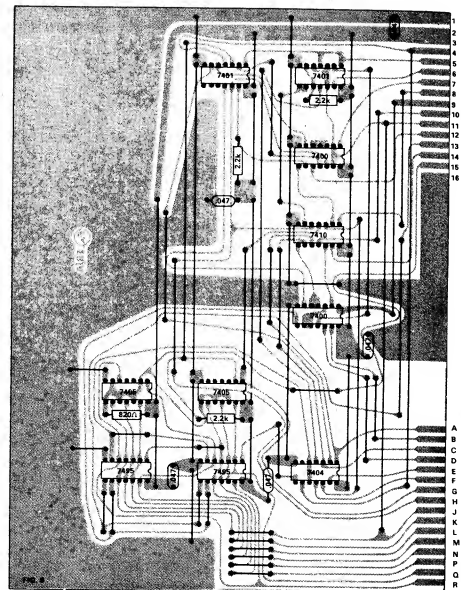
Hopefully, your machine will have passed all these tests, and you will be ready to work on the program counter and adder board, and the memory board. These will be described next, so that before long you will be able to get the machine actually running in its basic form.

## Troubleshooting

If you have struck trouble, in that your machine doesn't perform as it should, there are probably three likely causes. One is that you may have made a wiring error, such as an IC in the wrong position or around the wrong way, or a wire link in the wrong position. The second possibility is that you have a faulty connection in one of the PC edge connectors — in which case cleaning the board connector pads with a cloth soaked in methylated spirit may help, or failing this judicious bending of a guilty socket clip may be required.

The third possibility is a faulty IC, which does occasionally occur. Here the trick is to find the guilty device, of course, after which the remedy is obvious.

I myself have encountered only two faulty ICs, both of which were 7400 quad 2-input gate devices. In both cases only one of the four gates was faulty, but the symptoms were identical: the gate operated purely as



an inverter from one input, completely ignoring the other input. This suggests a broken bonding wire from the package pin to the chip, and it was perhaps significant that both devices were of the moulded plastic variety.

In both cases the effect of the fault in terms of machine operation was to cause the machine to perform an operation not only when it should, but at other times as well. In one case it skipped on the SMA microinstruction regardless of whether the AC was actually negative or not, for example.

In a nutshell, the approach to use when troubleshooting is to first study carefully the symptoms, noting exactly what is going wrong, and when (use the slow running mode to help spot timing). Then look carefully at the logic diagram, and you will often be able to narrow down the fault area quite closely. Finally, test out your theory about where the fault may lie, using a logic probe or a scope to analyse what is going on.

It takes a while to get the hang of this, but you'll find it will probably come to you faster than you may anticipate. At first, the logic circuits may seem bewilderingly complex, but in reality only a small part of the total circuit is generally involved at any one time.

## PLEASE NOTE

In the initial explanation of EDUC-8 operation, it was stated that with a cleared AC register, the effect of the CMA microinstruction is to give the AC a content of 1. This is of course wrong: it leaves the AC with all bits set to 1, equivalent to minus 1 in 2's complement notation.

It should also be noted that the RAR microinstruction only corresponds to division by two when the initial AC register content is an even number — i.e., with bit 0 zero. This is because the content of bit 0 moves to the bit 7 position following RAR.

In Fig. 6 of the same section, showing the basic organisation of the machine, the inverter shown after gate 14 should be before this gate. The inverter is used for the CMA microinstruction.

Finally, please note that the PC and MA registers have now both been enlarged from 5 to 8 bits, to cope with the 8-bit address words required for the 128/256 memory.

# The EDUC-8 computer: getting it going — at last!

Continuing with the construction of our unique digital computer project, the author describes here the program counter and adder section, and also the memory section. This completes the basic machine, which should now be capable of running in a limited way via the console controls and indicators.

by JAMIESON ROWE

Having completed the assembly and testing of the timing and run control, decoder, and accumulator sections, you should now be in a position to complete the basic machine by adding the program counter and adder section, and the vital memory section. These will be described here, together with details of how to check operation as you progress.

If you are keeping up with the description, all going well you will have by the end of this stage a complete basic computer. The only thing you are missing is an "introvert", capable of communicating with the outside world (i.e., you!) via the console switches and LED indicators. So that while it will be capable of running, it won't perhaps seem very spectacular or impressive. This will have to wait until we give it the ability to deal with input-output devices, and provide it with some of these to communicate with.

These preliminary comments are just to let you know where we're heading, and perhaps to encourage you if you were beginning to falter. With that done, I hope we can proceed.

For convenience, the program counter (PC) register and the serial adder circuitry are grouped together on a single plug-in board, coded E8/P. This board is the same size as the other plug-in boards described, 21.5 x 16cm, and has connector pads to mate with both a 32-way and a 16-way edge connector socket. It plugs into the sockets fourth from the top on the mother board, immediately above the accumulator board.

The logic for the program counter and adder sections is shown in Fig. 1. As you will perhaps have already noticed, the PC register itself is formed by two 7496 five-bit shift register devices, with only three flip-flops being used in the first device. This gives a total of eight bits, so that the PC is capable of handling the 8-bit address words needed for 256 memory locations.

The five parallel inputs of the 7496 which is fully used are connected to pads 24-28 inclusive of the board edge connector, and thence to SRO-SR4 of the switch register. Similarly the three active parallel inputs of the second 7496 are taken to SR5, SR6 and SR7, in this case via flying leads (this device has been added to the original design, as part of the memory expansion to 256 words, and this is the reason for the flying leads).

The connections between the eight parallel inputs and the switch register are for loading addresses into the PC.

The actual load address operation is

carried out by means of the LA1 (L) and LA2 control signals, which you may remember are generated on the front panel board by logic connected to the "load address" key. These signals arrive at the PC board via pads 17 and 18. The LA1 (L) signal is connected to the clear inputs of the 7496s, to clear them of any previous content. The LA2 signal line connects to the parallel load enable inputs (PL), and this signal thus causes the new address to be loaded from the SR after clearing.

The least significant bit (LSB) output of the PC is connected to the C-bus output line, pad 6, via a control gate. The control signal for this gate is produced in turn by a second gate, connected as a 2-input OR element. The two signals fed to this gate are the T2-9. (F OR DEP OR EXM) (L) signal fed to the board via pad 20, and a T2-9.JMS (L) signal derived from the T2-9 and JMS (L) signals which arrive at the board via pads 13 and D, respectively.

Thus the LSB output of the PC is connected to the C-bus and the B2 input of the serial adder during T2-9 of either a fetch, deposit or examine cycle, and during T2-9 of a JMS execute cycle.

Similarly the LSB output of the PC is also connected to the B-bus output line, pad 5, and the A1 input of the serial adder, by means of a control gate fed with a T14-21.SKIP signal derived from pad 22. Thus this pathway is enabled during T14-21 of the execute cycle of any of the instructions involving instruction skipping.

The serial input of the PC is permanently connected via an inverter element to the A-bus input, pad 4. The output of the serial adder is connected to the A-bus line via a control gate, whose control signal is produced by a 4-input gate functioning as an OR element. And among the signals fed to this element are the T2-9. (F OR DEP OR EXM) (L) signal, and indirectly via further gating the T14-21.SKIP (L) signal. These signals also fed to other gates connected to the control inputs Ac and Bc of the serial adder, which will be described in more detail in a moment.

MCP pulses arriving at the board via pad 11 are fed to the clock pulse inputs of the 7496 devices via yet another control gate, whose control signal is again produced by a 3-input OR element. And not surprisingly, the signals which thus enable the clock pulse gates are T2-9. (F OR DEP OR EXM) (L), T2-9.JMS (L), and T14-21.SKIP (L).

The net result of all this is that during T2-9 of a fetch, deposit, or examine cycle, or T14-21 of the examine cycle for a skip in-

struction, the contents of the PC are effectively shifted out of the register to the A-bus, through the adder, and back into the register again. Providing the carry FF of the adder is set to one before this operation, the PC content is therefore incremented at these times.

In addition, the content of the PC is also shifted out onto the C-bus during T2-9 of a JMS execute cycle, ready to be written into memory.

In tracing through these logic paths you probably also noticed that a path from the C-bus to the serial input of the PC, via the serial adder, is also enabled during T14-21 of either a JMS or JMP execute cycle. This is performed by the signal T14-21.JMS OR JMP (L), which arrives at the board via pad 21. This signal also enables the clock pulse gate of the PC.

The purpose of this further logic is to allow the content of the PC to be replaced as required during a JMS or JMP execute cycle, from the MA register.

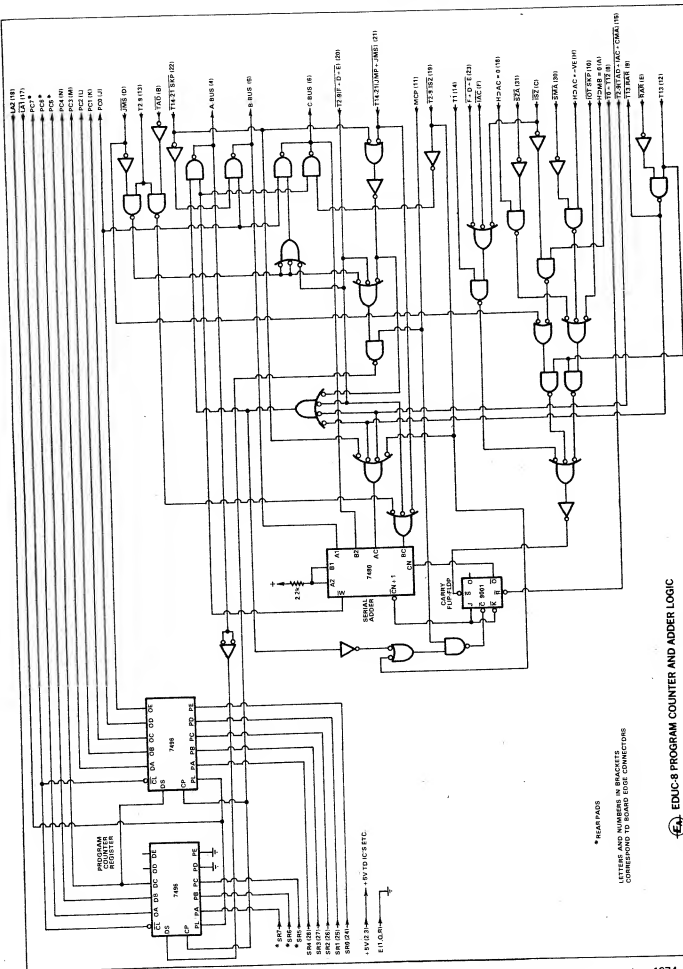
This completes the PC register logic, which as you can see is not very complex. The remaining logic on the board is that associated with those serial adder functions which do not involve the PC.

The serial adder consists of a 7480 full adder device, with a 9001 high-speed J-K flip-flop to store the carry. The adder inputs A1 and B2 connect to the B-bus and C-bus respectively, and are controlled by logic signals applied to gating inputs Ac and Bc respectively. The two unused inputs A2 and B1 are tied together and taken to the 5V rail via a protective resistor.

The output of the adder is taken to the A-bus via a control gate, as we have already noted. It is also taken to the C-bus via another control gate, fed with a T2-9.ISZ signal derived from pad 19. This is used for the "increment" part of the ISZ instruction.

The carry FF is reset at times T0 and T12 of every machine cycle by the (T0 OR T12) (L) signal applied to its R-bar input from pad 8. Where the adder is to be used for incrementing either the PC, the AC or a number fed from memory, the carry FF is then set to 1 at either of times T1 or T13, by means of the logic connected to its S-bar input. For brevity this will not be described in detail; but note, for example, that setting occurs at T1 for either fetch, deposit, examine, IAC or ISZ. The setting at T13 occurs for a variety of other signals and signal combinations, such as ISZ again, or SZA together with a clear accumulator.

When the adder is used for incrementing, the signal to be incremented is applied to either the A1 or B2 inputs, from the B-bus or C-bus, and appropriate control signals applied to the adder control inputs and the output control gates to enable the right paths. The same procedure occurs for true binary addition, which occurs only during T2-9 of the TAD execute cycle. In both cases MCP pulses are also fed to the clock input of carry FF, so that it correctly stores the individual bit carries.



\* REAR PADS  
LETTERS AND NUMBERS IN BRACKETS  
CORRESPOND TO BOARD EDGE CONNECTORS

EDUC-8 PROGRAM COUNTER AND ADDRESS LOGIC

## EDUC-8 computer

Apart from incrementing and addition, the adder is also used as a passive data path from the B-bus to the A-bus, during T2-9 of the CMA execute cycle, and at time T13 of the RAR execute cycle. It is similarly used as a data path from the C-bus to the A-bus, during T14-21 of the JMP execute cycle. The latter path is used to transfer the new "next instruction" address from the MA into the PC.

You may care to follow through the detailed logic paths involved in the operation of the adder for yourself, using the basic organisation diagram and cycle and instruction timing table given earlier as a guide.

The wiring of the program counter board is shown in Fig. 2. This board is coded EB/P, and like those plug-in boards already described measures 21.5 x 16 cm. There are 14 ICs on the board, together with a handful of minor components and some wire links. Wiring up the board should be fairly straightforward using the diagram as a guide; the main points to watch are that the ICs are correctly orientated and that the links are between the correct pads.

Note that there are connection pads near the 7496 device whose axis is parallel to the "back" of the board, for the attachment of flying leads to switch register switches SR5, SR6 and SR7, and to the LED indicator circuits for PC5, PC6 and PC7. Flying leads are needed here because this is the device which I have added to expand the PC for the larger memory; it has not been possible to provide for the connections via the plug-in connectors and the mother board.

The flying leads to the switch register should be fairly self evident. Of those for the LED indicators, the leads marked PC5 and PC6 simply go to the points on the top edge of the front panel board which are marked (PC5) and (PC6) in the previously given wiring diagram. The remaining lead, that marked PC7, will be dealt with shortly; it requires special treatment because the front panel board has no provision for an eighth PC indicator LED or driver.

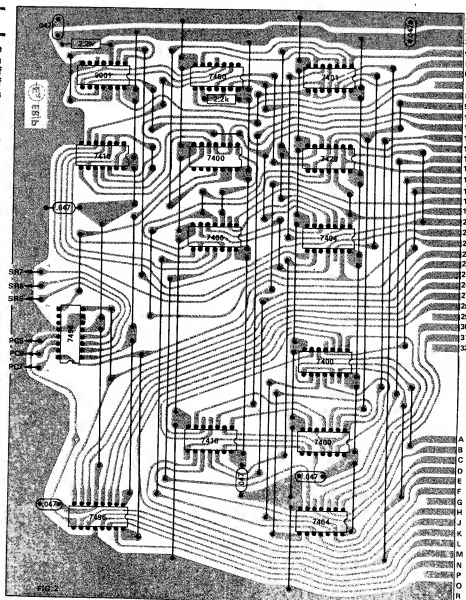
The logic circuit for the memory board is shown in Fig. 3. As you can see, this board includes the memory itself, the memory buffer (MB) register, and the memory address (MA) register.

The memory itself consists of two Fairchild type 93415 random-access memory (RAM) devices, although if only a 128-word memory is required, one device may be omitted. Both are needed for the full 256-word memory.

The 93415 device is actually a 1024-bit bipolar RAM, organised as 1024 single bits. However, for this application I have in effect "re-organised" the two devices so that they each store 128 x 8-bit words.

The technique used for this relies on the fact that EDUC-8 is basically a serial machine, and moves both instructions and data words around one bit after the other, in 8 clock pulse sequences.

The address number fed to the memory devices from the MA register is fed to the most significant address inputs of the 93415 devices, with the 8th and most significant bit used to determine which device is selected. This leaves the three least significant address bits of each device unspecified, so that the addresses provided by the MA basically correspond to groups of eight adjacent memory locations in either one device or the other.



The three least significant address bits of the memory devices, A0, A1 and A2, are connected together and also to the outputs of a 3-bit binary counter, formed by part of a 7493 device. This has been called the "memory strobe" counter, because its action is to cycle the total memory address applied to the two 93415 devices through the eight appropriate bit positions, during the eight MCP periods in which a word is either written in or read out.

In effect, the strobe counter and the decoding logic inside the 93415 devices for the three least significant address bits form an 8-way multiplexer and demultiplexer, which connects each of the locations in a group of 8 to the write or read circuitry in turn, during an 8-bit write or read sequence. The main "address" specified by the content of the MA simply determines which group of 8 bit locations is involved.

You can visualise this action as one whereby a number is stored by "spraying" its 8 bits into the 8 locations of the group selected by the MA address. Conversely, a number is read out serially by in effect "sniffing" each of the 8 locations at its address, in turn.

The MEMORY ENABLE signal arriving at pad 23 of the board (from the decoder board) is used to enable the memory devices

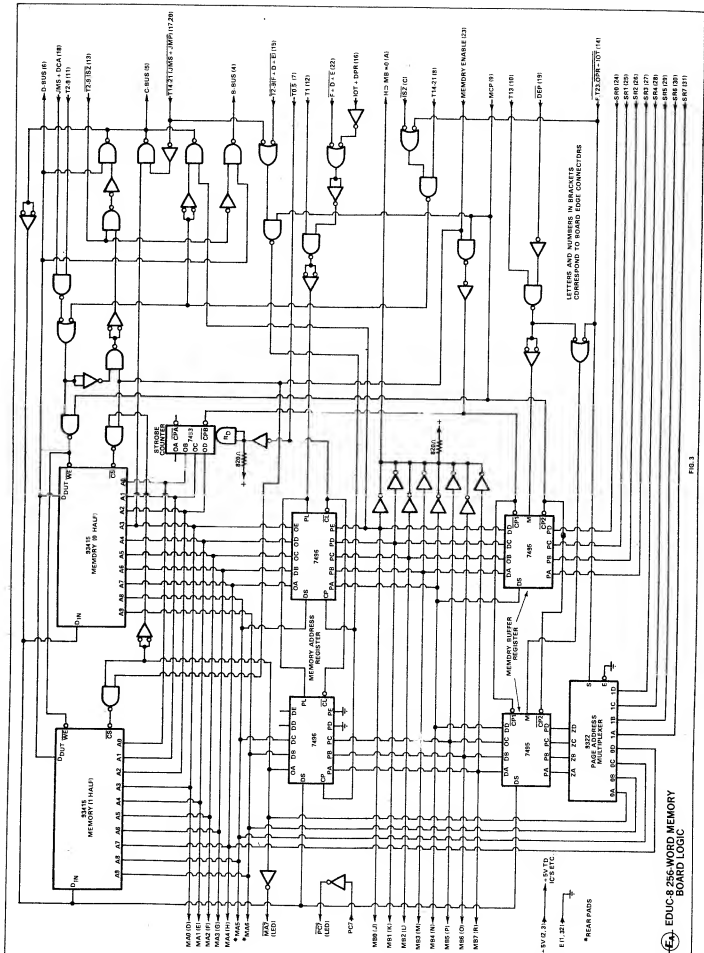
whenever they are involved in machine operation. The signal is fed to the "chip select" or CS-bar inputs of the 93415 devices via two gates, one fed with the MSB output from the MA register, and the other fed with the complement of this signal. Thus when bit 7 of the address is 0, only memory device "0" is enabled; conversely when bit 7 is 1, only memory device "1" is enabled.

The signal from pad 23 is also used to gate MCP pulses to the strobe counter and the MB register devices, so that both operate whenever the memory is enabled.

If the memory devices are enabled with the "write enable" or WE-bar control inputs held at the logic high level, they perform the read function. During the 8 MCP periods concerned the content of the 8 memory locations at the specified address are sensed in turn (non-destructively), and appear one after the other at the Do outputs of the devices, which connect to the D-bus (pad 6).

Further gating also provides a path from the Do outputs to the B-bus (pad 4) during T2-9 of an ISZ execute cycle, and to the C-bus (pad 5) for all read operations other than T2-9. ISZ.

To achieve the memory write function rather than read, the WE-bar control inputs



## For fine detail work — a hands free magnifier



The Magna-Sighter is a precision 3-D binocular magnifier that leaves your hands completely free for work. It has hundreds of applications, and is invaluable for scientists, technicians, craftsmen, toolmakers, hobbyists, etc. Slips easily over the head—over glasses, too. Proved and used by many U.S. universities, space research bureaux, government departments and major industrial organisations. Available in 3 different magnifications. Price \$18.00

## MAGNA-SIGHTER

For further information send this coupon today:

### STOTT TECHNICAL SERVICES ME/A574

(Division of Stott's  
Technical Correspondence College Pty. Ltd.)  
159 Flinders Lane, Melbourne, Vic., 3000

Please send me full information on the 3-D Magna-Sighter. I understand that no Sales Representative will call.

Name .....

Address .....

Postcode .....

STC 721

KENWOOD



# QR-666

the ALL-band

## COMMUNICATIONS RECEIVER

that gives  
you the world  
and an FM  
option, too.

All-band/all-mode reception on frequencies 170 kHz to 30 MHz covered by 6 bands. Receives broadcasts in any mode AM, SSB, CW or FM—with the optional accessory QRP-FM. Super sensitivity from dual gate MOS types FET's, double signal selectivity and AGC characteristics. IF circuit with mechanical and ceramic filters designed for high selectivity, resistance to interference; single button selection of wide band (5 kHz/8dB) or narrow band (2.5 kHz/6dB). Altogether a high performance compact, smartly styled unit of advanced design at a suggested "Today" price of \$332.20.



— Mail coupon NOW! —

Weston Electronics Company  
215 North Rocks Rd., North Rocks, N.S.W. 2151 Phone 630-7400

NAME .....

Please send details of  
the Kenwood QR-666

ADDRESS .....

POSTCODE .....

## EDUC-8 computer

of the 93415 device selected must be held at the logic low level during the time when the device is enabled. This is achieved by means of a "write" control signal, applied to the WE-bar inputs via a gate used to AND the signal with the MCP pulses. This is to restrict the actual writing operation to the second half of each clock period, to ensure that logic levels on data lines and address lines have all stabilised before writing commences.

The memory address or MA register is formed by two 7496 five-bit shift register devices, connected in very similar fashion to those for the PC register. Here, however, the parallel inputs of the devices are connected to the outputs of the MB register.

The clear (CL-bar) inputs are connected to edge controller pad 7, so that the MA is cleared at time T0.5 of every machine cycle.

There are only four actual situations when the memory write cycle is required: T2-9 of either a JMS or DCA execute cycle, or T14-21 of either a deposit cycle or an ISZ execute cycle. The "write" control signal is therefore generated by four gates fed by signals from pads 11, 18, 8, C and 19. The complement of the write control signal is also used to disable the data path between the Do outputs of the memory devices and the C-bus, as this bus line is used to feed in the data to be written.

The parallel load enable (PL) inputs of the 7496s are also connected together, and fed with a signal formed by a simple logic circuit combining signals from pads 12, 16 and 22. This causes the MA register to parallel load the number in the MB register at time T1 of either a defer cycle or an execute cycle involving any of the memory reference instructions — i.e., time T1 for all cycles other than fetch, deposit, examine, IOT execute and OPR execute.

The MA register operates as a shift-right register at times T2-9 of a fetch, deposit or examine cycle, and at times T14-21 of a JMS or JMP execute cycle. Clock pulses are fed to the CP inputs of the 7496 devices by the gates deriving signals from connector pads 9, 15, and 17. Note that the bit-0 output of the MA is also connected to the C-bus line, via a control gate fed with the T14-21 (JMS OR JMP) signal. This is used to feed the operand address from the MA to the PC via the adder, which either increments it in the case of a JMS cycle, or passes it unchanged in the case of a JMP cycle.

The memory buffer or MB register is formed by two 7495 four-bit shift registers, and in that respect is similar to the accumulator. The outputs of the 7495 devices are here also taken to eight inverter elements, wired as before to form an 8-input NOR gate. This is used to generate a "MB equals 0" signal, required for the conditional skip part of the ISZ instruction.

For most of the time, the MB register operates in tandem with the memory, storing and displaying the number being written into or read from a memory location. For these operations the MB operates as a simple shift-right register, and this is achieved by holding the 7495 mode control (M) inputs at low logic level, with clock pulses applied to the CP-bar inputs when appropriate.

There are two situations where this mode of operation is not used, and where the 7495 devices are made to accept parallel

data input. The first of these situations occurs during time T13 of a deposit cycle, when the parallel inputs of the MB are connected to the switch register switches, to load in the number to be deposited. A T13. DEP signal is derived from the signals on connector pads 10 and 19 for this purpose, and is used to drive the M inputs of the 7495 devices to logic high level for one MCP period. As MCP pulses are connected to the CP2-bar inputs of the devices, this causes loading to occur at time T13.5 of the deposit cycle.

Note that whereas the parallel inputs of the 7495 device correspond to the 4 least significant bits of the MB connect directly to SR0, SR1, SR2 and SR3, the parallel inputs of the second device connect to the remaining four switch register lines via a 9322 quad 2-input multiplexer device. This is because these inputs must alternatively connect to the outputs of the four most significant MA register bits, for the second MB parallel loading situation.

This situation occurs at time T23 of the fetch cycle, when a memory reference instruction has been fetched. Here, you may recall, the four most significant bits of the instruction address must be transferred from the MA to the MB, so that the machine will "remember" the page of memory from which it has fetched the instruction. It needs this information to complete the operand address, in the case of a direct memory reference instruction, or the address of the operand address in the case of an indirect instruction.

The switching (S-bar) input of the 9322 multiplexer is normally held at logic high level, and thus the "1" inputs of the device are connected to its outputs. The parallel inputs for the four most significant bits of the MB are thus normally connected to SR4, SR5, SR6 and SR7, and remain so during the deposit operation. However the S-bar input of the 9322 is taken to pad 14, which also feeds through an inverting OR gate to the M input of the coupled 7495. As a result, the arrival of the F.T23. not (OPR OR IOT) (L) signal at pad 14 causes the 9322 to switch the parallel inputs of the 7495 to the MA outputs, from the SR lines, and at the same time the 7495 M input is driven high so that loading takes place.

The wiring diagram for the memory board is shown in Fig. 4. Like the other plug-in boards this one also measures 21.5 x 16cm, being coded 58/M. As you can see, it involves 18 IC's, a small number of passive components, and again a number of wire links.

Wiring the board should present few if any problems if this diagram is followed carefully. As before the main points to watch are that the ICs are correctly oriented and in the correct positions, and that the wire links are wired correctly.

Note that the memory device positions are marked "0" and "1" to signify which device is designated by the appropriate values of bit 7 of the MA register. In other words, device "0" is that which forms the "first half" of the 256-word memory, and device "1" the second half.

I would suggest that you use sockets for the two memory devices, as they are by far the most expensive ICs in the whole machine. Use high quality sockets, preferably of the "low profile" type so that they do not raise the devices too high from the board. Otherwise you may have trouble with the device packages fouling the decoder board, when the board is plugged in.

Like the program counter board, the

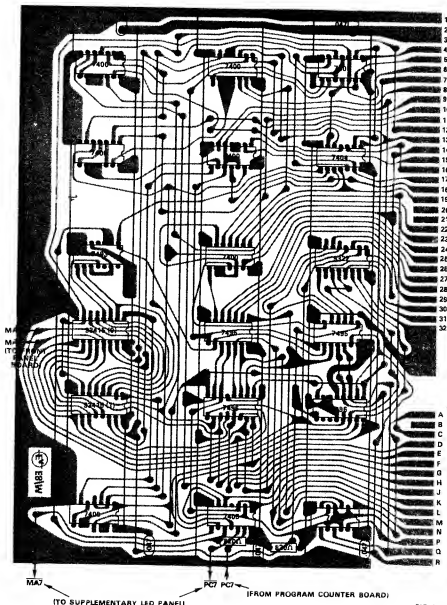


FIG 4

memory board has some flying leads connecting to it. As before these have been made necessary because of the expansion of the memory to 256 words.

The two flying leads marked MA5 and MA6 should be fairly self-explanatory. They go to the points on the top edge of the front panel board which are marked (MA5) and (MA6), to drive the appropriate LED indicators.

You may recall that the front panel board has no provision for the eighth indicator LEDs of the PC and MA registers, PC7 and MA7, or for their drivers. These LEDs must be mounted on a small supplementary panel, which is mounted in front of the front panel board so that the LEDs are correctly positioned in line with the others.

Two inverter elements on the memory board, which would otherwise have been unused, have been arranged to serve as the drivers for the two additional LEDs. The input of the inverter for the MA7 LED is connected to MA7 by the memory board pattern, and only its output need concern the constructor. This is the flying lead marked MA7-bar, which goes to the supplementary panel.

To allow the second inverter to be used as the PC7 driver, a lead must be wired between its input and the PC7 output on the program counter board. This is the flying lead marked PC7. The output of the driver then goes to the supplementary panel, like that for the MA7 LED, via the lead marked PC7-bar.

Details of the supplementary panel used to support the PC7 and MA7 LEDs and their series 180 ohm resistors are shown in Fig. 5. The board measures 21 x 32mm, and is cut from a scrap of Veroboard with 0.1in conductor spacing.

Only one conductor strip need be cut on the panel - that third from the top, used for terminating the two resistors and the flying leads to the memory board. Note that the resistors should be the small 1/4-watt size, to fit comfortably. The two LEDs should also be of a type having a fairly short body, and mounted head against the panel to produce a shallow overall assembly.

As shown, the panel is supported in front of the front-panel PC board in such a position that the two LEDs are in line with those in the rest of the array. The panel is supported by two small "U" shaped strips, fashioned from a scrap of 18G sheet brass

## EDUC-8 computer

or similar. As well as supporting the panel, the strips also complete the 5V supply connection for the LED anodes, as they are soldered to the 5V copper area on the front panel board.

With both the program counter and memory boards completed and checked over carefully, you should now be ready to try them out. The first step is to plug in the program counter board, after having carefully cleaned its edge connector pads and made sure that it mates correctly with its sockets. It plugs into the mother board in the fourth position down, don't forget, immediately above the accumulator board.

Now turn on the power. Things should be much the same as with the first three boards in position, except that now the LED indicators for the PC register should show some random number instead of being all lit (apart from PC7, which has only just been added).

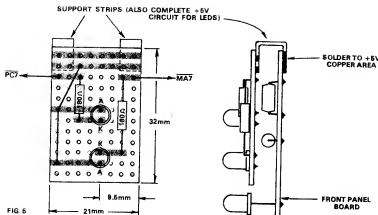
If all is well so far, set the SR switches all to zero (down position) and press down the load address key. All PC indicator LEDs should go out, and should stay out when the key is released.

Now try loading an address into the PC, by setting the SR switches to a number other than zero, and repeating the process. The LEDs should remain dark when the load address key is pressed, but adopt the bit pattern set by the SR switches when it is released.

Try a variety of numbers in this way, just to make sure that all is well. Note that when the load address key is pressed, the previous content of the PC should be cleared, with the new content being loaded in only when the key is released.

You can now test the PC incrementing function. To do this, set the FAST / SLOW switch to the lower position (fast), and the SINGLE / CONT switch to the upper position (single). Then press the examine key, whereupon the number in the PC should increment — increase by one. Further pressing of the examine key should repeat the process, and in fact you should be able to run the PC right through its full 256 bit combinations, by simply pressing the key enough times.

Pressing the deposit key instead of the examine key should have the same effect at this stage, so try this also. And you can also try pressing the run key, which should have the same effect again, at least in terms of



the PC being incremented. It may also have other effects, depending upon the setting of the SR switches (which will still be doubling for the MB register, assuming you have left in the temporary connections used in the previous tests).

You can now check the "skip" incrementing of the PC, or at least one instance of it — the SZA microinstruction. Do this in the following way. First, set the SR switches to octal 710, corresponding to CLA. Then, leaving the FAST / SLOW and SINGLE / CONT in their present positions, press the run key. This should clear the AC.

Now set the SR switches to octal 730, corresponding to SZA, and note the number in the PC register. Then press the run key again, and you should see the PC increment. Further presses of the run key should produce further increments.

If all is well this far, your PC is working as it should, and the serial adder must also be fairly right. However to test the adder further, you can run through the various OPR microinstructions, and check that they are working correctly.

Perhaps the best one to check first is CMA, whose octal code is 704. So set the SR switches to this, and press the run key. This should cause the AC content to change from all zeros to all ones. Further presses of the key should simply cause the AC to swing back and forth between these two situations.

You have already tried the SZA microinstruction to check that it causes PC incrementing with an AC content of zero, but just to make sure that all is well, end up the CMA test with the AC set to all ones. Then set the SR switches to octal 730 again, and try pressing the run key. This

time the PC should not increment.

Now set the SR switches to octal 701, corresponding to IAC. Pressing the run key should now cause the AC to clear, as the "all ones" content corresponds to minus 1 in 2's complement notation, and minus 1 incremented gives zero. Further presses of the run key should cause further increments, so that the AC content should become 1, 2, 3, 4 and so on.

Now try testing the RAR microinstruction, by setting the SR switches to octal 722. Pressing the run key this time should cause the number in the AC to be rotated to the right by one bit, with the content of bit 0 transferred to bit 7. Each time you press the run key the number in the AC should move by one bit in this way, so that a total of 8 presses should restore the number to its original position.

If you stop the number with a 1 in the bit 7 (most significant bit) position, this will let you try the SMA microinstruction. Do this by setting the SR switches to octal 724, whereupon pressing the run key should cause the PC register to be incremented. Do this a few times, just to make sure that incrementing occurs each time. Then set the SR switches to octal 702 (RAL) or 722 (RAR), and rotate the number in the AC until there is a 0 in the bit 7 position. Now reset the SR switches to 724, and press the run key once more. This time the PC should not be incremented, as the AC content is no longer "negative" (according to 2's complement notation).

The combined OPR microinstructions can also be tested at this stage, although if all has been well so far, they should all work as a matter of course.

Thus CLA, IAC (octal 711) should cause the AC to be set to 1, while CLA, CMA (octal 714) should cause it to be set to minus 1 instead. Similarly CMA, IAC (octal 705) should cause the number in the AC to be changed into its 2's complement, while SZA, SMA (octal 734) should cause the PC to be incremented if the number in the AC is either zero or negative.

Having now tested just about all of the functions of the machine in its incomplete form, the next step is the big one: adding the memory board, to complete the basic machine and make it capable of running.

Prepare for this by turning off the power, and then removing the temporary connections between the SR switches and the MB register outputs.

Note that if you have only one 9341S memory device at this stage, it would be best to plug it into the "0" socket. The discussion which follows will in fact assume that this is the case.

Clean the edge connector pads on the

## EDUC-8 COMPUTER PARTS LIST — 3

### PROGRAM COUNTER & ADDER BOARD

- 1 PC board, code E8/P, 21.5 x 16cm
- 5 7400 or 9002 quad 2-input gate IC
- 1 7401 or 9012 quad 2-input gate with open collectors
- 2 7404 or 9016 hex inverter IC
- 2 7410 or 9003 triple 3-input gate IC
- 1 7480 or 9380 gated full adder IC
- 2 7495 or 9395 5-bit shift register
- 1 9001 high speed J-K flip-flop
- 2 2.2k  $\frac{1}{4}$ W resistor
- 5 0.047uF LV polyester or ceramic insulated hookup wire for links, flying leads.

### MEMORY BOARD

- 1 PC board, code E8/M, 21.5 x 16cm
- 6 7400 or 9002 quad 2-input gate IC
- 1 7401 or 9012 quad 2-input gate with open collectors
- 2 7405 or 9016 hex inverter
- 2 7405 or 9017 hex inverter with open collectors
- 1 7493 or 9393 4-bit counter
- 2 7495 or 9395 4-bit shift register
- 2 7496 or 9396 5-bit shift register
- 1 9322 quad 2-input multiplexer
- 2 9341S 1024-bit RAM (or only 1, for 128-word memory)
- 2 820 ohm  $\frac{1}{4}$ W resistors
- 5 .047uF LV polyester or ceramic 16-pin DIL sockets. Low profile high quality type
- Insulated hookup wire for links, flying leads.



## EDUC-8 computer

memory board carefully, and plug it into the sockets on the mother board between the program counter and decoder boards. Then turn on the power once more.

Set the SR switches to zero (all down), and load this into the PC by pressing the load address key. Then, with the SLOW / FAST switch set to the down or fast position, press the examine key. A random number should appear in the MB register, representing the turn-on bias bits of the flip-flops in the first 8 bit locations of the 93415 device in the "0" memory socket.

Press the examine key a few more times. This should bring out more random numbers into the MB, probably a different number each time (but not necessarily).

Now you can try loading in a few numbers, using the deposit function. To do this, first load a suitable starting address into the PC, by setting it up on the SR switches and pressing the load address key. You can load in any convenient starting address you fancy, although an easily remembered one is the very first: location zero.

After having loaded the starting address into the PC, set up a suitable number on the SR switches. Then press the deposit key, and you should see the number appear on the MB registers LEDs. At the same time the starting address should have transferred from the PC to the MA, while the PC content should have incremented.

If all seems well, set up another number on the SR switches, and press the deposit key again. The process should be repeated, with the new number appearing in the MB. Note that you do not have to load in a new address for the second deposit, because the PC has already incremented after the first. You can deposit a third and fourth number (or more) if you like by further deposits.

By these steps, you should have stored the numbers into a consecutive group of memory locations. To check that this has in fact happened, set up your initial starting address again on the SR switches, and load it into the PC with the load address key. Then press the examine key, and the number you stored in the first address should appear in the MB again. Further presses of the examine key should cause the second, third, fourth and other numbers stored to appear also.

If all is well so far, the odds are that your memory is working correctly, and the machine should now be capable of running. So probably the best thing to do next is to load in a simple program, and see if it runs.

The simple test program shown is probably a good one to try, as it is quite short and easily loaded. Don't worry too much about the program itself, as we will deal with programming soon. At this stage all you need to know is that it is a very simple one, using mainly the ISZ and JMP instructions, which causes the accumulator to be incremented 256 times.

First set up a suitable starting address on the SR, and load it into the PC. Then deposit each of the eleven octal code numbers of the program in turn, as before. After this has been done, check that you haven't made any mistakes by loading the starting address into the PC once again, and examining the stored numbers. If they check out correctly against the list, you are just about ready.

The final step in preparation is to move the SINGLE/CONT switch to the down position, to allow the machine to run



## EDUC-8 PROGRAM

### "FIRST TEST"

STEP	MNEMONIC	CODE
0	START, CLA	710
1	INCR, IAC	701
2	BACK, NOP	700
3	NOP	700
4	ISZ INDX	211
5	JMP BACK	502
6	ISZ INDY	212
7	JMP INCR	501
10	HLT	721
11	INDX, 0	000
12	INDY, 0	000

*A simple program you can use to check that your machine is capable of running. It merely increments the AC register 256 times, and then stops.*

continuously. The FAST/SLOW switch should already be in the down position, for fast operation, and this is correct.

Are you ready? All you have to do now is load the starting address once more into the PC, and press the run key.

Upon doing so, the RUN indicator should light, with the FETCH and EXECUTE indicators also lighting a little less brightly to indicate that the machine is flitting back and forth between them. The ISZ, JMP and OPR indicators should also be partially lit, showing that the machine is doing these types of instruction.

But more dramatic than these should be the AC register LEDs, which if all is going well will be showing a brisk counting operation. The binary number in the AC should be incrementing, probably at a rate just too fast for you to keep track. The indicators for the PC, MA and MB registers should all be partially lit, perhaps with some brighter than others, indicating that these registers are involved in a lot of dynamic activity.

In the discussion of progress testing given at the end of the description of the timing, decoder and accumulator boards, it was stated that the CMA microinstruction could be tested with only these plug-in boards in position. In fact this is not so, as the CMA recirculation loop uses the serial adder as its return pathway. So if you tried this test and it didn't work, don't worry!

The CMA microinstruction will work correctly when the program counter board is now added, however, as described in the present discussion.

If you have had trouble in finding a reservoir capacitor for the power supply, please note that 33,000µF 25VW units are readily available from Siemens Industries. Order via your supplier.

This situation should continue for about 30 seconds, until the accumulator fills up and overflows. The machine should then promptly stop running.

If you used location zero as your starting address, as shown, the registers should now show the following octal contents:

PC: 011 MB:721  
MA: 010 AC:000

If this is the case, you can be fairly sure that your machine is working correctly. At this stage we have not tested some of the memory reference instructions, but this can wait. For the moment, you will no doubt want to try feeding the test program into other parts of the memory, and make sure that it runs there also.

You can also try setting the FAST/SLOW switch to the upper or slow position, and try running the program at the slow clock rate. This will enable you to follow its operation in detail, as it fetches out each instruction and then executes it.

But please note one important point: when depositing or examining, ALWAYS make sure that the FAST / SLOW switch is set to the down or fast position. Otherwise, the machine will not correctly recognise that a deposit or examine cycle is required, and will start running. This will do no harm, but can be annoying!

Incidentally, you don't have to feed in the test program again simply in order to re-run it; merely load the starting address into the PC as before and press the run key. That is all, unless of course you want to store it in another part of the memory, and run it there. In this case, you will have to feed it in again.

Your machine should now be ready for the addition of the sixth plug-in board, the input-output transfer or IOT interface board. This will be described next, along with a simple input keyboard device and a simple output display — your first two "peripherals".

# EDUC-8: adding the input/output interface

If you've been keeping up with the description of our computer project, by this stage you're probably eager to add the final section, so that it will be able to interact with peripherals. Details of this section are given here, together with a simple input keyboard unit and a low-cost output display unit.

by JAMIESON ROWE

The remaining section of the basic machine to be described is the IOT interfacing logic, which handles programmed transfer of data between the machine itself and any input and output "peripherals" which may be connected to it. This logic is mounted on the last plug-in PC board, which plugs into the lowest position on the mother board. Coded EB / IOT, the board measures 16 x 21.5cm like the boards previously described.

The logic diagram for this section of the machine is shown in Fig. 1. As may be seen, it is the simplest and most straightforward section in the machine, involving a relatively small number of gates and inverters.

You may recall that the design of the machine is such that it will interface with a total of four input-output devices at any one time; two input devices and two output devices. These connect to the machine via four rear-panel sockets, which for convenience are labelled "Input device 0" (ID0), "Input device 1" (ID1), "Output device 0" (OD0), and "Output device 1" (OD1).

Broadly speaking, it is the task of the IOT interfacing logic to select the device designated by an IOT instruction, and perform one or more of the three basic operations involving that device. These are testing the device's flag, transferring data to or from the device, and resetting the flag.

Generally they are performed in that order, although this is not necessarily the case.

From earlier discussion you may recall that bits 3 and 4 of an IOT instruction are used to specify the device concerned. Bit 4 is used to differentiate between input and output devices, while bit 3 is used to indicate either a "0" device or a "1" device.

Inverted versions of the signals corresponding to these two instruction bits, derived from MB3 and MB4 of the memory buffer register, reach the IOT board via edge connector pads 5 and 6. The inverters and gates connected to these pads form a simple one-of-four decoder, whose outputs are each used to gate one of the four sets of input-output device logic. Thus only one of the latter sets can be operative at any one time, corresponding to the device specified in the instruction. This part of the logic thus acts as a "device selector".

As you have probably already noticed by now, each device interface involves four logic signals: flag sensing (L), flag reset (L), test clock pulses (L), and data (L for output devices). Of these the first is always in effect an "input" signal, the second and third are always "output" signals, and the last is an input signal for input devices, but an output signal for output devices.

The flag sensing inputs for all four device interfaces use negative logic polarity (true equals L). This has been done to ensure that

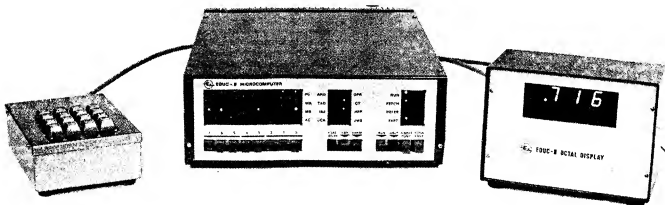
if a selected device interface has no device actually connected, a program will be able to detect the condition, and not proceed in error. An unconnected TTL input floats to high level, so that with negative logic the flag sensing input of an interface floats to the "flag not set" state if no device is plugged into the corresponding socket. Hence the absence of the device makes itself apparent to a program in terms of a continuously "reset" flag condition.

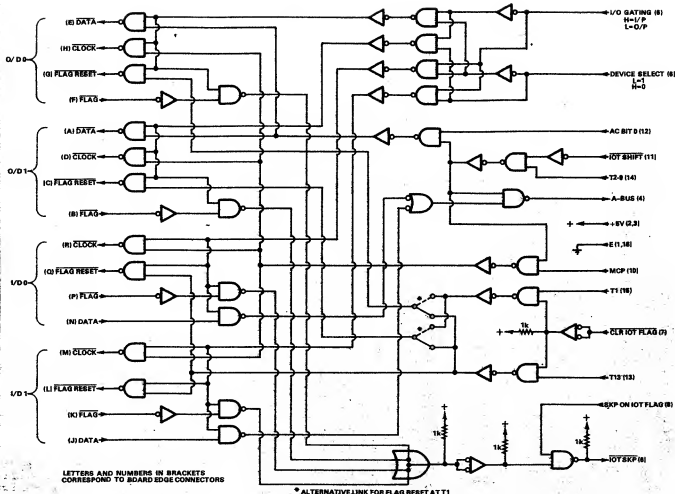
Upon entering the interface, the flag sensing signals are inverted and gated with the appropriate device select signals from the decoder. The gates used are open-collector types, and the outputs of all four gates are combined to achieve a wired-OR function. The resultant signal is then inverted and gated with the "SKP on IOT FLAG" signal from the main instruction decoder, which enters the board via edge connector pad 8. The output of the gate thus goes low if, and only if, there is a "skip on IOT flag" instruction, and the selected device flag is set. The gate output, labelled IOT SKP (L), is taken to edge connector pad 9, from where it goes to the program counter and address board.

A signal to reset input device flags at time T13 of a "clear IOT flag" instruction is generated by inverting the CLR IOT FLG (L) signal entering the board via pad 7, and using this to gate the T13 timing signal which enters via pad 13. This is then inverted and gated by the device selector signals as before, to produce the final FLAG RESET (L) signal for each device.

The clr iot flag.t13 signal can also be used to reset the output device flags, in the same way, and this is in fact the normal way of connecting the output device flag reset logic. However to allow for possible situations where a particular output device may require its flag to be reset before data transfer, rather than after, provision has been made for an alternative CLR IOT FLAG.T1 signal to be used. This is

*Picture below shows the basic machine together with the two simple peripheral devices described in the present section. At left is the simple keyboard unit, while at right is the octal display unit.*





EDUC-8 IOT INTERFACE LOGIC

FIG. 1

generated in the same way as the first signal, using the T1 timing signal from pad 15.

Either reset signal may be used for each of the two output device interfaces, simply by fitting the appropriate wire link in one of two possible positions.

The actual data transfer between peripherals and the machine takes place during times T2-9 of an IOT shift instruction. Accordingly an IOT SHIFT, T2-9 signal is produced by taking the IOT SHIFT (L) signal entering the board at pad 11, inverting it, using it to gate the T2-9 timing signal entering at pad 14, and inverting the resultant. This is then used for three tasks, the first of which is to gate master clock pulses (MCP) entering the board at pad 10. After inversion the resultant signal is fed to each interface for gating with the appropriate device select signal to produce each CLOCK (L) output signal.

The second application of the IOT SHIFT, T2-9 signal is to gate the AC BIT 0 signal entering via pad 12, to enable the output data path from the accumulator register. After inversion the output of the gate is fed to each output device interface for further gating by the device select signals, to produce each DATA (L) output. Thus during an IOT shift instruction specifying an output device, the device selected is able to receive the 8 data bits from the AC register during times T2-9 of the instruction execute cycle.

Finally, the IOT SHIFT, T2-9 signal is also used to enable a data path between the input device interfaces and the machine A-bus data line. Each input device data line is gated by the appropriate device select signal, as before, then the two are combined in a gate performing the OR function. The output of this gate is then gated by the IOT SHIFT, T2-9 signal, and the resultant fed to the A-bus via pad 4. Hence if an IOT shift instruction specifies an input device, the 8 data bits from that device are able to pass to the A-bus (and ultimately to the AC register) during times T2-9 of the instruction execute cycle.

Only eleven low-cost ICs are used on the IOT interface board to perform these functions, and the wiring is quite straightforward. The diagram of Fig. 2 shows the position and orientation of the ICs on the board, together with the position of the wire interconnection links and the few minor components used.

The diagram shows the output device flag reset signal links in the "T13" positions, with the alternative link positions for "T1" resetting shown dashed. I suggest you wire the board initially with the links in the T13 positions, as shown, as this is likely to be suitable for most output devices you will want to use. One or both links can always be changed over at some later stage, if you find this necessary for a particular peripheral. The link furthest from the edge

connector is that for OD0, while the other is that for OD1.

When the board is wired, and you are confident that no errors have been made, clean its edge connector pads with a soft cloth moistened with methylated spirit, and plug it into the lowest socket position on the mother board. All going well, your EDUC-8 microcomputer should now be complete, and potentially capable of "conversing" with the outside world via peripheral devices.

There is an almost endless variety of devices to which a micro-computer like EDUC-8 can be connected. On the input side, almost any piece of equipment whose "output" or status can be encoded as an 8-bit binary number is a potential input device. Similarly, any piece of equipment whose operation is capable of being controlled or "programmed" by an 8-bit binary number, or which is capable of accepting an input signal digitally encoded as a stream of such numbers, is a potential output device.

Don't just assume, then, that the only possible peripherals for your EDUC-8 are the conventional "attachements" one associates with a traditional computer — like a paper tape or card reader, a punch or a line printer. These have their uses, and because you will no doubt want to know how they can be hooked up to your machine, I will try and give details for as many of them as possible in following sections. But the field

## EDUC-8 computer

is wide open for you yourself to experiment with all sorts of ideas, using the computer to monitor and control the operation of anything that takes your fancy.

Only by people experimenting in this way will the applications of computers be extended, and their full potential be realised. So once you have your EDUC-8 machine operational and you have a few simple peripherals under your belt, don't be shy! Let your imagination loose, and see if you can't come up with a completely new computer application.

So that you will have a couple of simple peripherals to "cut your teeth on", as it were, I have produced a very simple and low cost input keyboard unit, and a companion octal display output unit. These are very basic input and output units, but they are easy to build and should serve to illustrate the basic principles involved. Details of the two units are given in the remainder of this section.

The keyboard unit is a very simple unit using only five ICs together with an array of low-cost silicon diodes for encoding. A sixth IC may be added if desired, to drive LED indicators for showing the buffer register contents and flag status.

The heart of the unit is a 16-key keyboard assembly made by Mechanical Enterprises, Inc., of Virginia, and available in Australia from General Electronic Services Pty Ltd. The catalog number of the keyboard is type SK 760, and it is fitted with mercury keyswitches — which feature bounceless contacting. The keyboard is available with a set of double-shot moulded keytops, with the inscriptions shown in the photographs.

The logic for the keyboard unit is shown in Fig. 3, except for the diode encoding array. As may be seen, a 7496/9396 five-bit shift register is used for the buffer register, to keep things simple and minimise costs. Although the register has a basic capacity of 5 bits, it is used to generate 8-bit words by using a little bit of "trickery". The additional 3 bits are produced by manipulating the logic level applied to the serial data input  $D_s$ .

As the encoding table shows, the encoding used is a slightly modified version of the standard known as "ASCII" (American Standard for Computer Information Interchange). The ten numerals are encoded simply as their binary equivalent, while the other symbols are encoded as the binary

equivalents of decimal 10, 11, 12, 13 and 14, except that the three most significant bits are also set.

The key inscribed "CTRL" is used to control the value of the fifth bit in the encoded word. Its action is thus rather like a shift key on a typewriter, enabling each of the other 15 keys to be used to produce a key unit is able to produce 30 different output characters, making it quite flexible.

The diode encoding matrix uses negative encoding, so that the parallel loading inputs of the 7496 register are normally at high logic level. When any of the keys except the CTRL key are pressed, at least one of the four least significant bit lines are taken low. The diode encoding matrix uses negative encoding, so that the parallel loading inputs of the 7496 register are normally at high logic level. When any of the keys except the CTRL key are pressed, at least one of the four least significant bit lines are taken low. The diode encoding matrix uses negative encoding, so that the parallel loading inputs of the 7496 register are normally at high logic level. When any of the keys except the CTRL key are pressed, at least one of the four least significant bit lines are taken low. The diode encoding matrix uses negative encoding, so that the parallel loading inputs of the 7496 register are normally at high logic level. When any of the keys except the CTRL key are pressed, at least one of the four least significant bit lines are taken low.

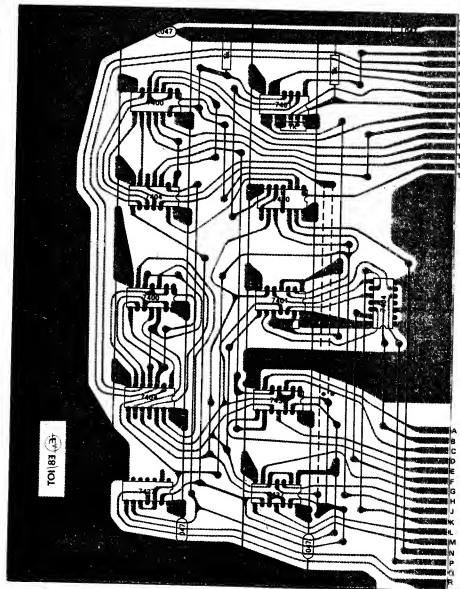
The trailing edge of the one-shot output pulse is also used to trigger a second one-shot, formed by the other half of the 9602. This produces a further pulse (negative

logic), which is used to set the flag flip-flop. The flag FF is a simple R-S type formed from two cross-coupled gates. Its output is buffered by a 7437 gate element connected as an inverting driver, to become the flag output line of the unit.

The FLAT SET (L) signal appearing on this line thus indicates to the computer that a character has been received by the keyboard, and is available. As soon as the program in the computer tests the flag line and senses that it has been set, it accordingly sends a set of eight shift clock pulses to the keyboard along the SHIFT CLOCK (L) line, to enable the character to be shifted out of the keyboard and into the cor AC register.

The eight data bits leave the buffer via the E output, and pass through two 7437 gates connected as buffer elements. They then go along the DATA output line to the computer IOT interface. Note that the shift clock pulses reaching the keyboard from the computer interface are also passed through a 7437 buffer element, which acts as a line receiver and pulse restorer.

When the data has been transferred to the AC register of the computer, the last phase of the IOT transfer takes place. The IOT interface sends a RESET FLAG (L) pulse to the keyboard unit, and this is used to reset



SIMPLE KEYBOARD — ENCODING

Key	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	0	1	0
LF	1	0	1	0	1	0	1	0	1	0	1	0	1
+	1	0	1	0	1	0	1	0	1	0	1	0	1
X	1	0	1	0	1	0	1	0	1	0	1	0	1
-	1	0	1	0	1	0	1	0	1	0	1	0	1
+	1	0	1	0	1	0	1	0	1	0	1	0	1
CTRL	1	0	1	0	1	0	1	0	1	0	1	0	1

the flag FF. It is also used to clear the buffer register, making sure that the latter is ready to receive a new character.

As you can see, it is quite an elementary input unit, using a bare minimum of logic. At the same time, it provides all that is really necessary, at a fairly low cost. The main cost will be the keyboard assembly itself. This will involve a little more outlay than for similar units using mechanical keyswitches, but the zero bounce of the mercury switches allows the logic to be kept simple.

I used some of the flat bonded multi-wire cable from the wiring between the keyboard and the logic board, as you can see from the photograph. This makes the job a little easier, because of the colour coded wires, and also makes the final result more attractive.

I have produced a small PC board pattern for the keyboard. This is coded EB / K1, and measures 15.2 x 10.2cm. The wiring for the board is shown in Fig. 4; as you can see, the board includes the wiring for the diode encoding array. The connector pads for the interconnection wires linking the diodes to the keyboard output pads, to simplify the wiring (see Fig. 5). This assumes, of course, that you fit the keytops to the switches in the positions shown. Otherwise you will have to rearrange the connections.

Although space is provided on the board for the 7405 IC, this is not required for basic operation of the keyboard unit. It is only needed if you want to provide the unit with a set of LED indicators to show the contents of the data buffer and the status of the flag FF. The additional wiring is shown in Fig. 3 using dashed lines.

If you decide to add this facility, note that no provision has been made on the PC board for mounting either the LEDs or their 180-ohm series dropping resistors. The six

LEDs would be mounted on, or behind a small window in the case front panel, with the resistors on a small tagstrip nearby.

The works of the keyboard unit are housed in a small utility box. I used one of the hammettone finish cases from Wardrobe and Carroll Fabrications Pty Ltd, measuring 14 x 11.5 x 5cm and having a wrap-around lid. The logic board is mounted in the bottom of the case, using four screws which are also used to attach rubber feet to the case underside. Nuts are used to space the board up from the case, to prevent shorts.

The keyboard assembly is screwed to the underside of the case lid, using 16.5mm long specers to ensure that the keys protrude by the appropriate amount. The

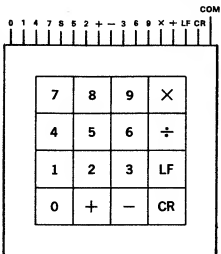
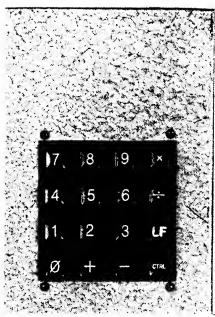


FIG. 5



This top view of the simple keyboard shows the keypad inscriptions and positions. The connections for the switch assembly are shown at left. A full alphanumeric keyboard may be described at a later stage.

clearance hole in the lid for the keys is square, and measures 76mm per side.

The keyboard unit receives its 5V power from the computer, via the same cable used to make the logic connections. The cable used must therefore provide a minimum of six conductors. I used a multi-wire cable of the type made for TV remote control units, and this particular cable has six insulated wires together with a single shielded wire. I used the shielded wire for the data line, with the shield braid connected in parallel with one of the other wires as the earth line. The spare unshielded wire was connected in parallel with the active 5V line, to hopefully lower its impedance also.

The cable enters the keyboard unit case through a grommeted hole, and is clamped to prevent strain on the connections. The wires are then separated and connected directly to the PC board pads. The other end of the cable is fitted with a 6-pin DIN plug, with the connections corresponding to the IOT socket connections given in an earlier section.

Before we leave the simple keyboard unit, there are a couple of minor points which should be borne in mind when the unit is in operation. Because of the simple logic used, the unit cannot store the three most significant data bits of the output character prior to the actual data transfer. This means that for correct encoding from the non-numeric keys (except CTRL), these keys must be kept pressed at least until three of the eight shift clock pulses have occurred.

When the computer is operating at its fast clock rate, this is a rather academic point, because the computer will typically shift the character out of the keyboard within about 300µs of the flag being set. It is unlikely that you will be able to press a key down for less than this!

However if the fast / slow switch on the computer is set to the slow rate, to make it easier to analyse operation, you will have to remember to hold the non-numerical keys down until the character is shifted out. Otherwise, the three most significant bits

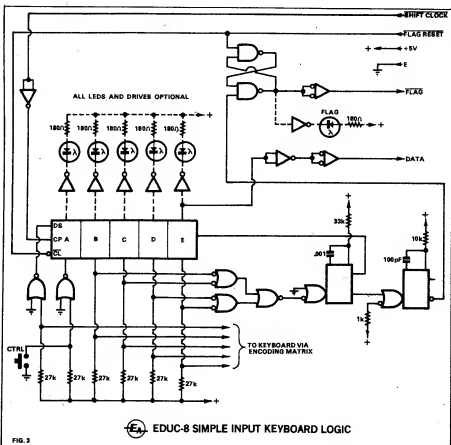


FIG. 3

EDUC-8 SIMPLE INPUT KEYBOARD LOGIC

## EDUC-8 computer

will not be anecdotal. This is a minor point, but one which should be remembered.

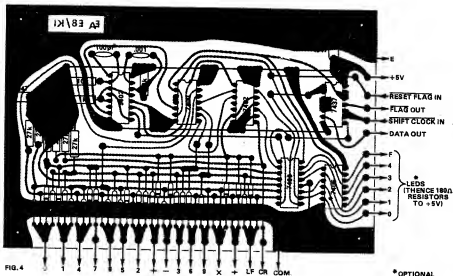
The second point to note is that the CTRL key does not itself initiate loading of the keyboard buffer, or setting of the flag FF. Pressed alone, it will have no effect. Rather, it must be pressed in conjunction with one of the other keys. The correct procedure is to press the CTRL key first, hold it down, and then press the desired active key. The two are best released in the reverse order — active key first, then the CTRL key.

From the keyboard unit let us now turn to the simple display unit. Like the keyboard, this has been designed to demonstrate the basic essentials of a practical peripheral — in this case an output device.

In broad terms, the display unit simply receives an 8-bit data number from the computer and displays it as an equivalent 3-digit octal number. The octal digits are displayed on three 7-segment LED readouts, and the particular LEDs used are a fairly new type from Litronic having brighter and larger (0.6in high) digits than usual.

The reason why octal digits are used for the display, rather than decimal, is that conversion from an 8-bit pure binary number to the equivalent decimal digits is not easy. In comparison, conversion to the equivalent octal digits is quite straightforward, since it is merely a matter of decoding each group of three bits independently.

Although the fact that the display reads out in octal may seem a disadvantage, in fact it is quite useful since the instructions for a machine like EDUC-8 are most conveniently handled in octal. In many situations it is convenient to think of the data in terms of octal notation, also. From a tutorial viewpoint, the exercise and mental flexibility involved in translating from octal into more familiar decimal can also be very worthwhile.



To encode three full octal digits, nine bits would be required, and as we have in this case only 8 bits, this means that the display is really only one of "2 1/2" digits. To make it more flexible, I have provided the display with a switch, giving two alternative decoding formats. One has the partial digit in the most significant position, i.e., the "377" format, while the other has it in the centre position, giving the "737" format.

The first format is in some ways the logical one, and is the one you would normally use for displaying data numbers. However the second format corresponds to that used for the EDUC-8 instructions, which use the three most significant bits for the operation code. By providing the display with a switch, you can readily select the format most suitable for what is being displayed.

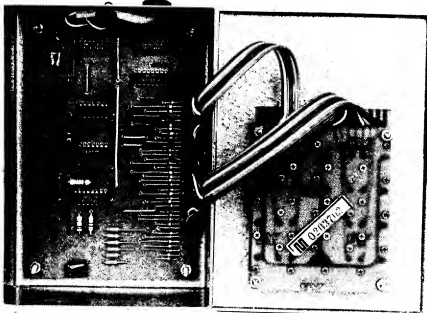
The logic for the display unit is shown in Fig. 6. As you can see, it is again fairly straightforward. Data from the computer passes through a 7437 gate acting as a line

receiver and reshaper, and then enters the buffer register. This is a 74164 / 93164 8-bit shift register device, very handy for this sort of application.

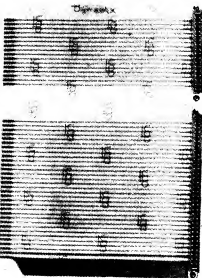
The eight outputs of the buffer register are then connected, some via the format selector switch, to three 9307 decoder devices. These are normally designated as decimal to 7-segment decoders, but by grounding the A3 inputs they become octal to 7-segment decoders. Driver transistors are then used to match the 9307 outputs to the inputs of the three Litronic type DL750 readouts, due to the voltage drop of the double-junction LED segments.

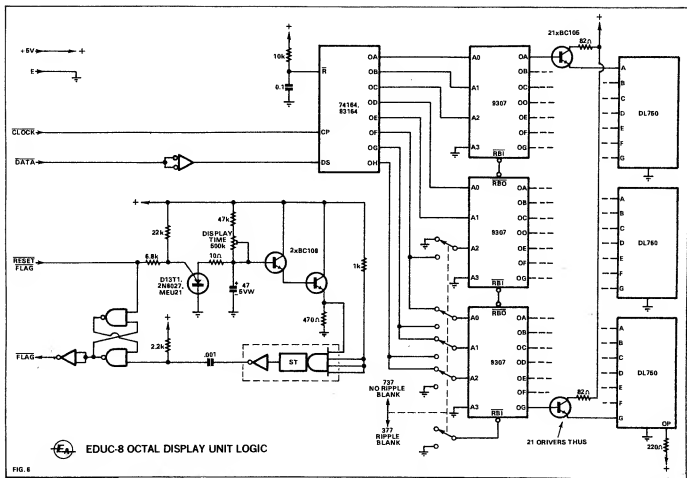
The DL750 readouts should be available through your normal supplier, on order from the Australian agents for Litronic, Cema Distributors Pty Ltd.

As well as changing the decoding connections, the format switch is also used to control the leading zero blanking facility provided by the 9307 decoders. One switch pole is used to enable the blanking for 377 format readout, and inhibit it for 737 format. Leading zeroes are thus blanked when the display is being used for data words, but unblanked for display of instruction words — where all bit combinations have a



Above is a view of the interior of the simple keyboard unit, showing the switch assembly and the PC board complete with diode encoding matrix. At right is the extender board used for servicing the computer itself.





significance and must be displayed.

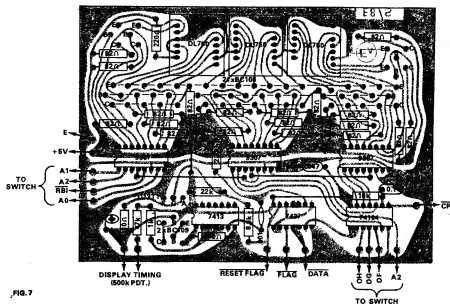
The R-C network connected to the reset (R-bar) input of the 74164 buffer is used to automatically clear the buffer when the 5V supply is connected to the unit. This gives the display its own "turn-on reset" facility, preventing spurious displays even if the unit is plugged into the computer with power applied.

This does mean, however, that if the format selector switch is in the 377 position, so that leading zero blanking is operative, the display would normally be completely blank when power is applied. To ensure that it is always easy to check that the display unit is "active", with power applied, the "decimal point" LED on the DL750 for the most significant digit is connected via a 220-ohm resistor to the 5V rail. It thus acts as a pilot lamp.

Strictly speaking, because the display unit does not have to perform any time-consuming mechanical or electrical processing of the 8-bit numbers fed to it, there is little need for it to indicate its status to the computer via a flag system. After all, in itself it is capable of displaying numbers just as rapidly as the computer is capable of shooting them out!

However, from the human user's point of view, each number displayed should last a reasonable time, to allow it to be recognised and perhaps recorded on paper before it is succeeded by the next number. Because of this requirement, then, rather than from a need to allow time for its internal processing, the display unit is provided with a flag system.

As before, the flag flip-flop is formed from two gates from a 7437 IC, with a third gate used as a buffer-driver for the flag (L)



output line back to the computer. The remaining circuitry is simply a time delay, arranged so that after the computer sends the display a number and resets the flag, a predetermined time elapses and then the flag is set again to indicate that the display has been held for a suitable time.

The display unit itself can thus be used to regulate the rate at which numbers are fed to it. This can simplify programs, as a time-delaying instruction loop might otherwise be necessary to regulate the display rate.

However note that the computer and its programme are under no obligation to test flag status before sending another number; as the flag system is an arbitrary one, it can be ignored if desired. In this respect the display is rather different from most peripherals.

The actual delay circuit used is one which is quite simple, yet allows stable and easily adjusted delays. The PUT is triggered by the RESET FLAG (L) pulse from the computer, discharging the 47uF capacitor, and turning off the two Darlingtons transistors. The input

KENWOOD



QR-666

the ALL-band

COMMUNICATIONS RECEIVER

that gives  
you the world  
and an FM  
option, too.

All-band/all-mode reception on frequencies 170 kHz to 30 MHz covered by 6 bands. Receives broadcasts in any mode AM, SSB, CW or FM—with the optional accessory GRS-FM. Super sensitivity from dual gate MOS types FET's, double signal selectivity and AGC characteristics. IF circuit with mechanical and ceramic filters designed for high selectivity, resistance to interference, single button selection of wide band (5 kHz/6dB) or narrow band (2.5 kHz/6dB). Altogether a high performance compact, smartly styled unit of advanced design at a suggested 'Today' price of \$336.20.



—Mail coupon NOW!—

Western Electronics Company  
215 North Rocks Rd., North Rocks, N.S.W. 2151 Phone 630-7400

NAME \_\_\_\_\_

Please send details of  
the Kenwood QR-666

ADDRESS \_\_\_\_\_

POSTCODE \_\_\_\_\_

## NOTICE TO ADVERTISERS

The Trade Practices Act 1974 came into force on October 1, 1974.

Certain provisions of the Act relating to consumer protection place a heavy burden upon advertisers, advertising agents and publishers of advertisements.

Section 52 of the Act imposes a general duty on everyone (individual and corporation alike) not to engage, in trade or commerce, in conduct that is "misleading or deceptive."

In addition Section 53 (read with Sections 6 (3) (c) and 79) makes it a criminal offence (punishable in the case of an individual by a fine of \$10,000 or 8 months' imprisonment and in the case of a corporation by a fine of \$50,000) for an individual or corporation to do any of the following in trade or commerce in connection with the supply or possible supply of goods or services or in connection with the promotion by any means (for example advertising) of the supply or use of goods or services, namely:—

- (a) falsely represent that goods or services are of a particular standard, quality or grade, or that goods are of a particular style or model;
- (b) falsely represent that goods are new;
- (c) represent that goods or services have sponsorship, approval, performance characteristics, accessories, uses or benefits they do not have;
- (d) represent that the individual or

corporation has a sponsorship, approval or affiliation [he, she or] it does not have;

- (e) make false or misleading statements concerning the existence of, or amounts of, price reductions;
- (f) make false or misleading statements concerning the need for any goods, services, replacements or repairs; or
- (g) make false or misleading statements concerning the existence or effect of any warranty or guarantee."

Apart from the criminal sanction for a breach of Section 53, an individual or corporation infringing Section 52 or 53 is liable to proceedings for injunction and for damages suffered by an injured party. In view of the obvious impossibility of our ensuring that advertisements submitted for publication comply with the Act, advertisers, and advertising agents will appreciate the absolute need themselves to ensure that the provisions of the Act, including the sections specified above, are complied with strictly. It is suggested that in cases of doubt advertisers and advertising agents seek legal advice.

### SUNGRAVURE PTY. LTD.

## EDUC-8 PARTS LIST — 4

### IOT INTERFACE BOARD

- 1 PC board, code EB/10T, 16 x 21.5cm
- 3 7400 or 9002 quad 2-input gate IC
- 2 7401 or 9012 quad 2-input gate with open collectors
- 3 7404 or 9016 hex inverter IC
- 3 7437 quad 2-input buffer IC
- 4 1k  $\frac{1}{4}$ W resistors
- 4 .047  $\mu$ F LV polyester or ceramic capacitors
- Insulated hookup wire for links, etc.

### SIMPLE KEYBOARD UNIT (OPTIONAL)

- 1 PC board, code EB/K1, 152 x 101mm
- 1 7496/3396 5-bit register IC
- 1 7400 or 9002 quad 2-input gate IC
- 1 7402 quad 2-input positive NOR gate IC
- 1 7437 quad 2-input buffer IC
- 1 9602 dual one-shot IC
- 37 General purpose silicon diodes, type 1N914, AN2003, etc.
- 1 1k  $\frac{1}{4}$ W resistor
- 1 10k  $\frac{1}{4}$ W resistor
- 6 27k  $\frac{1}{4}$ W resistors
- 1 33k  $\frac{1}{4}$ W resistor
- 1 100 $\mu$ F polystyrene or ceramic
- 1 1000 $\mu$ F polystyrene or ceramic
- 2 .047 $\mu$ F LV polyester or ceramic
- 1 Metal case (see text)
- 1 16-switch keyboard, Mechanical Enterprises type SK760, with set of keytops as described
- 1 6-pin DIN plug
- Length of cable for interconnection (see text)
- Rubber feet, screws, nuts, 16.5mm spacers, etc.

Required only for

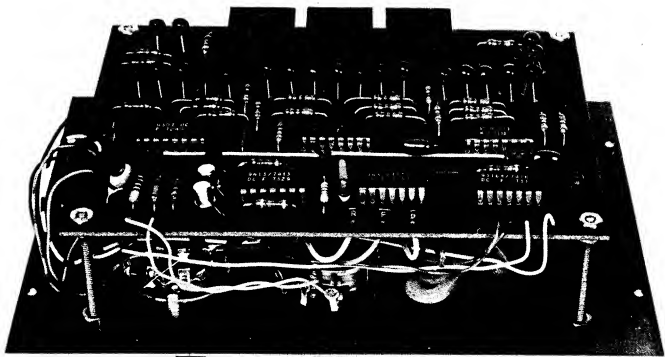
LED indicator facility:

- 1 7405 or 9017 hex inverter with open collectors
- 6 LEDs, type OLD419, FLV110, 5082-4850, CQY24 or similar.
- 6 180-ohm  $\frac{1}{4}$ W resistors

### OCIAL DISPLAY UNIT (OPTIONAL)

- 1 PC board, code EB/S, 140 x 101mm
- 3 DL750 7-segment LED displays
- 3 9307 decoder ICs
- 1 74164/93164 8-bit register IC
- 1 7437 quad 2-input buffer IC
- 1 7413 Schmitt trigger IC
- 21 8C108 or similar NPN silicon transistor
- 2 8C109 or similar NPN silicon transistor
- 1 D1371, 2N6027 or MEU21 PUT
- $\frac{1}{4}$ W resistors: 1 x 10-ohm, 21 x 82-ohm, 1 x 220-ohm, 1 x 470-ohm, 1 x 1k, 1 x 2.2k, 1 x 6.8k, 1 x 10k, 1 x 22k, 1 x 47k
- 1 500k linear pot
- 1 100 $\mu$ F polyester or ceramic
- 2 .047 $\mu$ F LV polyester or ceramic
- 1 0.1 $\mu$ F LV polyester or ceramic
- 1 47 $\mu$ F 6VW tantalum electro
- 1 Metal case (see text)
- 1 5-pole 2-position miniature rotary switch
- 1 6-pin DIN plug
- Length of interconnecting cable (see text), 4 rubber feet, piece of orange perspex for viewing window, 2 knobs for controls, grommet, screws, etc.





At right is a view of the completed octal display, while above is a close-up of the PC board mounted on the back panel.

of the 7413 Schmitt trigger is thus taken low, and its output switches to the high state. This condition remains until the 47 $\mu$ F capacitor re-charges, through the 47k resistor and 500k pot. As soon as the capacitor voltage rises to two V<sub>be</sub> drops above the Schmitt trigger threshold, the trigger switches and its output drops to the low state. The .001 $\mu$ F coupling capacitor thereupon feeds a negative-going pulse to the flag FF, to set it.

The 500k pot may be used to adjust the time delay from a minimum of about 2 seconds to a maximum of around 20 seconds. When the computer program uses the flag system of the display to regulate the display rate, the pot therefore becomes the display time edjstmant.

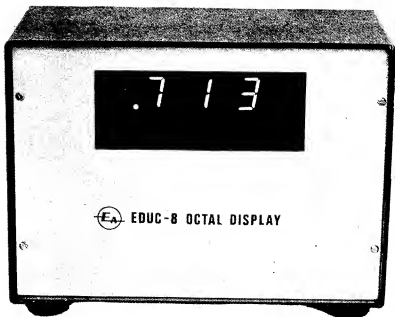
As with the simple keyboard unit I have produced a PC board pattern for the octal display. The board is coded EB/S, and measures 14 x 10.2cm. The wiring diagram for the board is shown in Fig. 7. The only parts of the display unit which do not mount on the board are the format switch and the 500k timing pot.

The construction of the display unit should be fairly clear from the photographs. I built the prototype into a vinyl-covered steel case made by the Australian Transistor Company. Coded type 70-50-40, it measures 191 x 127 x 102mm, and has a brushed aluminium front panel.

The PC board is mounted on the removable rear panel of the case, using 2-inch long screws and nuts as spacers. The format selector switch and display time pot

#### PLEASE NOTE

The components for the mother board, shown in PARTS LIST 1, should show 2 x 470 ohm resistors, not 2 x 4.7k.



are also mounted on the rear panel, underneath the board, while the connection cable enters the case via a grommetted hole and is clamped there also. This keeps the construction very simple, as all the "works" are etched to the rear panel and can be removed as an assembly for easy access.

The case front panel has a rectangular cutout 10 x 4cm, opposite the readouts, with a piece of orange-tinted perspex cemented inside to improve viewing contrast. In this way the case itself acts as a viewing hood for the display, and they are clearly seen from a distance of many metres.

Four rubber feet on the bottom of the case complete the unit itself.

As with the simple keyboard, the display unit obtains its 5V power from the com-

puter, and must therefore connect to the latter via a cable having at least six conductors. I used a length of the same "TV remote control" cable as before, and again used the shielded wire for the DATA line. The cable again terminates in a 6-pin DIN plug, to mate with one of the EDUC-8 IOT device sockets.

Well, you now have the information to complete your EDUC-8 microcomputer, together with the details of two simple peripheral units to connect to it. More advanced peripherals will be described and discussed shortly. But before we deal with more hardware, some discussion of basic programming would no doubt be appreciated, and this will be the subject of the next section.

# Programming your EDUC-8 microcomputer

At this stage you will no doubt be keen to try your hand at writing some programs, so that you can really start making your computer jump through hoops. With this in mind, we take a break here from talking about hardware, and discuss basic programming.

by JAMIESON ROWE

Computer programming is a big subject, and in the space available here we will only be able to scrape the surface. We will deal only with basic machine language programming, as it affects a simple machine like our EDUC-8. Even so, it will be necessary to assume that you have a fairly good grasp of the basic operation of the machine given in an earlier section. If this material has become a little hazy, I would suggest that you read through it again before proceeding.

To recapitulate briefly, it is worth stressing again that a digital computer like EDUC-8 deals only with binary numbers. This applies to both the data upon which it operates, and the instructions which specify its operations. In fact the only difference between data and instruction numbers is that the latter are interpreted as such, a point which should become clearer as we go on.

In the case of EDUC-8, the binary numbers used for both data and instructions are of eight bits. Any eight bit number is capable of being interpreted by the machine as an instruction, in the sense that all combinations of eight bits have a significance when interpreted as an instruction.

For example the binary number 00000000 corresponds to zero when it is interpreted as a data number, but if interpreted by EDUC-8 as an instruction it causes the machine to perform a logical AND operation between the number in the accumulator register (AC) and the number stored in the memory location whose address in 0000 in the current memory page—that 16-word portion of the memory in which the instruction itself is stored.

In the same way, the number 11111111 is equivalent to decimal 255 in simple binary, or minus 1 in two's complement binary. But if interpreted by EDUC-8 as an instruction, it would cause the machine to increment the program counter if the number in the AC register is either zero or negative, at the same time rotate the number in the AC register one bit to the right, and finally halt the program.

As noted earlier, there are eight basic types of instruction "trick" in the machine's repertoire, with some of these having a variable form determined by the location in memory of the operand to which they refer, and others being subdivided into specific sub-tricks or micro-instructions. Every different micro-instruction and instruction form has its own specific 8-bit binary code number.

In the trust and most basic sense, then, a computer program is a sequence of binary numbers, in this case each of 8 bits. The computer itself is quite incapable of interpreting

instructions in any other form. A program written in binary number form is thus said to be in machine language.

Unfortunately, we human beings who must write the programs do not find it particularly easy to remember all of the binary code numbers corresponding to the various instructions. For convenience, then, it is usual for programmers to visualise and write machine language programs in octal notation. This gives code numbers which are quite readily remembered and manipulated, after a little practice.

For example an instruction whose binary form is `010101101` becomes 315 in octal form, while another whose binary form might be `11110010` becomes 722. You can see from these how much easier machine language programming becomes by using octal notation.

It is important to realise, however, that if programs are written in octal notation, or in any other non-binary form, this is done purely for the convenience of the human programmer. Computers themselves "understand" only binary numbers, and regardless of their initial form, all programs must ultimately be fed into the machine in binary.

A program written in octal notation is somewhat easier to handle than in binary, but still tends to be rather abstract and inconvenient from the human viewpoint. There is no obvious functional correspondence between the code numbers and the instructions they represent, making it necessary for them to be learned by rote. Even when this is done, it is by no means easy to visualise the operation of a program simply by scanning the code numbers.

It is for this reason that programs are often written in what is called "mnemonic language". Here each type of instruction is represented by a three or four letter symbol, whose form is arranged to make its significance easily remem-

bered. Hence the symbol "AND" is used to represent the logical AND instruction, for example, while "JMP" is used to represent the jump instruction.

Other easily remembered symbols are used to represent variations in the form of instructions, and as labels for memory addresses. For example an instruction might be written as "TAD I POINTN", where TAD stands for two's complement addition, I indicates that the instruction involves indirect addressing, and POINTN is the label given to the address in the current memory page containing the address of the operand.

So that you can start to become familiar with the octal coding and mnemonic symbols for the instructions in EDUC-8's repertoire, I have drawn them up as a table. This can be used as a convenient guide when writing programs, until you get to the stage where you know them all by heart.

By writing a program initially in mnemonic language, it is relatively easy for the programmer to visualise its operation. Of course it is still necessary to translate the program into binary coding for the machine, but this is not difficult if handled in octal. After a while, you'll be able to set the SR switches of the machine in binary, from octal coding, without batting an eyelid!

Needless to say, the translation from mnemonics into code is rather tedious, and the ideal solution is to have the computer do the job itself. In fact this is always done with full-scale machines, where a program known as an "assembler" is supplied as part of the software package sold with the machine, for this very purpose.

The idea is that the assembler program is stored in the machine, and under its control the machine reads the symbolic version of the new program—say from punched paper tape—and translates it to produce the binary code equivalent. This may be punched out as a second paper tape, known as the "object" tape. It is the object tape which is then used to feed the new program into the machine, when it itself is to be run.

Programs written in mnemonic form are often said to be written in "assembly language", to emphasise that they are one stage removed

## Recommended for further reading

Of necessity, the discussion of basic computer programming given here is only a brief introduction to the subject. It covers the basic principles, to a degree which should enable you to begin writing simple programs with a fair amount of confidence and success. But there will inevitably be questions raised in your mind which will remain unanswered. For a more complete introduction to the subject, I can only suggest that you refer to a modern textbook on the subject.

A book I can warmly recommend is "Introduction to Programming", published by the Digital Equipment Corporation. It is available from Digital Equipment Australia Pty Ltd, at a cost of \$2.50 including post and packing.

To obtain a copy, send a cheque for the above amount to the Education Manager, Digital Equipment Australia Pty Ltd, 123 Willoughby Road, Crows Nest 2065, with the envelope marked "Electronics Australia Enquiry".

from true machine language. But it should be noted that there is a simple 1:1 relationship between the two, in the sense that for every machine language instruction to be performed ultimately by the machine, there must be a corresponding assembly language instruction.

As you are probably aware, this process of using the computer itself to simplify programming and reduce the tedium is often carried a stage further. By providing the machine with a more elaborate translation program known as a "compiler", it can be made to translate programs written in more abstract language. The compiler program can be arranged to generate whole sequences of machine language instructions in response to a single input command, freeing the programmer from the need to worry about every tedious detail.

You have no doubt come across the names of the more abstract or "higher level" programming languages which have been developed to take advantage of compiler translation: FORTRAN, ALGOL, COBOL, BASIC, FOCAL, and so on. Because these programming languages make it particularly easy to write problem-solving programs, and have them running rapidly (after compiling), they are often called problem-orientated programming languages.

As you might expect, a compiler program tends to be quite long; very much longer than could be fitted into the 256-word memory of EDUC-8. This means that the convenience of writing programs in a problem-orientated language is simply not available with this machine, unless you care to compile programs manually.

On the other hand it may well be possible to write an assembler program small enough to fit into the machine's memory, to permit automatic assembly of programs written in mnemonics. I have not had time to try writing such a program as yet, but hope to do so soon. Details will certainly be published, if this proves practical.

In the meantime, I suggest that you write your programs in mnemonic form, and then manually code them in octal using the guide table. This is not difficult, and will be good practice. With the programs in octal form, for the time being you can feed them into the machine manually via the console switches. Later on, when you perhaps have a paper tape punch and reader, you can turn them into binary tapes for rapid and convenient loading.

Before we turn our attention to the actual "nitty gritty" of programming, a few broad comments on techniques are probably in order.

The first step in programming is to make sure that you define clearly the task which the computer is to perform. This may sound trite, but it is not. It is very important, for unless the task is clearly defined, it is all too easy to produce a program which may perform a task other than the one you really wanted. As it may be hard or tedious to modify it later on, the best way to avoid a lot of wasted effort is to make sure of defining the goal in the first place.

Having defined the job to be done, the next step is to decide upon the way in which it can be achieved. This step often tends to blend with the third step, which is that of analysing the task and breaking it down into the specific computer operations which will be necessary.

A very useful technique which can be used to simplify these steps is flowcharting. This involves drawing a graph or flowchart, which shows the various steps which will make up the program, and the logical sequence in which they are performed. By letting you visualise more clearly the steps involved, the flowchart makes it easier to refine and simplify the program before you progress any further. It also

## EDUC-8 PROGRAM ENCODING GUIDE

Mnemonic	Operation	Code
<b>MEMORY REFERENCE INSTRUCTIONS</b>		
AND	logical AND	0XX
TAD	2's complement add	1XX
ISZ	increment and skip if zero	2XX
DCA	deposit and clear AC	3XX
JMS	jump to subroutine	4XX
JMP	jump	5XX

(XX = operand address and mode)

### OPERATE (OPR) MICROINSTRUCTIONS

NOP	no operation	700
IAC	increment AC	701
RAL	rotate AC one bit left	702
CMA	complement AC	704
CLA	clear AC	710
NOP	no operation	720
HLT	halt at end of execute cycle	721
RAR	rotate AC one bit right	722
SMA	skip on minus AC	724
SZA	skip on zero AC	730

### COMBINED OPR MICROINSTRUCTIONS

CLA,IAC	set AC to contain 1	711
CLA,CMA	set AC to contain -1	714
SZA,SMA	skip if AC is zero or minus complement and increment AC (form 2's complement)	734
CMA,IAC		705

### INPUT/OUTPUT TRANSFER (IOT) INSTRUCTIONS

SKF	skip on input flag	601,611
SDF	skip on output flag	621,631
KRS	read input buffer	602,612
LDS	load output buffer	622,632
RKF	reset input flag	604,614
RDF	reset output flag	624,634

### COMBINED IOT INSTRUCTIONS

KRB	read input buffer, reset flag	606,616
LDB	load output buffer, reset flag	626,636

makes it possible to compare various approaches in tackling the problem (there are usually a number of possible ways, and it may not be easy to pick the most efficient).

After using flowcharts to settle upon the approach and refine the exact way in which the steps are to be performed, the next phase

is to actually write the program. This is generally known as the "coding" phase.

You might imagine that this would be the last real stage in the process of programming, followed only by assembly and storage in the machine before operation. However this is rarely the case. Generally there is a further stage, because human fallibility almost always ensures that a program won't work in its initial form.

The final stage is therefore one in which you perform "debugging". This involves running the program one or more times, noting the errors it makes, analysing these with the flowcharts and the written version of the program, and making the appropriate modifications to remedy matters. The modified assembly language version is then assembled once more, to produce the final object tape (hopefully!).

Let us now turn to the detail of programming. Probably the best way to start is with a very elementary example.

Suppose we have two numbers, stored in memory locations, and we wish to add one to the other and store their sum in a third memory location. This is a very simple task, and would not normally involve a program of its own; it would simply form a minor step in a larger program. However for the sake of the exercise, a flow-chart for the steps involved is shown in Fig. 1.

This is a very simple flow-chart, as you can see. There are only two different sorts of operation symbol, and the logical flow is in a simple linear fashion between the rounded START and HALT terminations. The rectangular boxes represent the functional steps, with the arrows

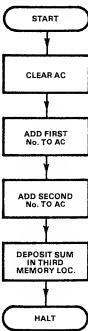


FIG. 1

## EDUC-8 computer

on the connecting lines showing the direction of flow.

The reasoning behind the various steps should be fairly self-evident. The accumulator register is first cleared, to make sure that there is no residue from a previous operation to confuse the issue. Then the first number is added into the accumulator, followed by the second number. This forms their sum in the accumulator, which is deposited in the desired memory location. Finally the machine is halted.

The mnemonic or assembly language form of a simple program to perform this task might look like this:

```
CLA      /start; clears AC
TAD A    /adds first number
TAD B    /adds second number
DCA C    /deposits sum
HLT      /halts
A,       /location of first number
B,       /location of second number
C,       /location in which sum is stored.
```

The first thing to note is that the actual program itself consists only of the left-hand column. All of the words to the right of the oblique slashes are comments, purely for the benefit of anyone trying to follow what is going on—including perhaps the programmer himself, at some later stage. Whether you add such comments to your own programs is entirely optional; a few at strategic points in a program can be very helpful, but it would normally be both unnecessary and tedious to put as many as shown here. I have added them merely to help you in getting the idea.

When a program written in assembly language is translated into binary code, these comments are completely ignored. Assembler programs may be arranged to do this automatically, by ignoring all characters on the symbolic type which follow a slash and precede a carriage return.

The other main thing to note about our first program example is that the three memory locations used to store the two numbers and the sum have been given labels—"A", "B" and "C". These are simple one-letter labels, but in practice these can generally be any convenient combination of letters and numerals—as long as they do not coincide with the mnemonics used for the actual instructions. This is not so important when you are coding programs yourself, manually, but it is essential if they are going to be coded by an assembler. Any ambiguity between labels and instruction mnemonics would then lead to errors.

The main advantage of using labels is that it frees you from worrying about the exact locations of each instruction and data number in memory, at least during the initial stages. It is still necessary to keep track of your position in memory, so as to be able to satisfy the requirements for memory reference instructions. But by using the labels you can leave the exact details of memory location until last, when the actual coding is done. And if the coding is done by an assembler program you may never need to worry about the details, as the program may do it for you!

Normally if the two numbers to be added together were to be loaded into the machine as part of the program, their numerical values would be shown after the commas following their address labels. On the other hand if they were going to be loaded separately, the initial content of the two locations might be shown

as zero. Similarly the content of the sum storage location would normally be shown as zero, to emphasise that the initial content is not significant. (But the latter is not necessary for correct operation of the program, as storing the sum in this location would automatically erase any previous content.)

Some of these points may become clearer if we look at the octal code equivalent to our program example. If the program were going to be stored in the computer's memory starting at the very first location, i.e., address 000, its coding would be as follows, where the locations in memory have been shown at the left for reference:

Location	Code
000	710
001	105
002	106
003	307
004	721
005	(first number)
006	(second number)
007	(sum stored here)

I suggest you spend a little time comparing this with the assembly language form, as this is probably the best way of grasping some of the ideas of coding. Note that the program occupies eight memory locations, five for the actual instructions and three for the storage locations. The two numbers to be added together would be stored along with the instruction numbers, in locations 005 and 006, and after the program has been run their sum will be found in location 007.

If you look carefully at the coding for the three memory reference instructions, some of the points made earlier about the details of memory location should start to become clearer. Because they must refer to the location of the operand concerned, the exact coding for such instructions varies with the position of the operand in memory. This is in contrast with the operate microinstructions, whose coding remains fixed.

To emphasise this, here is how the coding for the same program would look if we were to store it in the next group of eight memory locations, at the same time changing the positions of the three data storage locations so that they precede the instructions instead of follow them:

Location	Code
010	(first number)
011	(second number)
012	(sum stored here)
013	710
014	110
015	111
016	312
017	721

Notice how the exact form of the memory reference instructions has changed, to correspond to the new addresses of the operands, while the operate microinstructions are unaltered. The fact that the data storage locations are now "ahead" of the actual instructions has only altered the exact coding of the program, however, not its operation. It is still an entirely valid coding for the program task we wished to perform.

In practice it is often convenient to place at least some of the data storage locations before the actual instructions. The only thing to watch when this is done is that you remember to always start the program at the first actual instruction. If this is not done, the machine will fetch data numbers and interpret them as instructions — which can produce some rather strange results!

I hope this very elementary programming

example has helped you to grasp some of the rather subtle concepts involved. If nothing else, you will hopefully have begun to see by now the way in which instructions and data numbers stored in the machine differ only in terms of interpretation.

Let us now consider a programming example that is a little more complex. Say we have two numbers stored in the machine's memory, as before, but this time we wish to find the difference between the two and store it again—but always as a positive number. This time the flowchart would be as shown in Fig. 2.

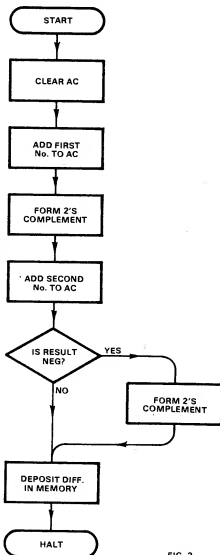


FIG. 2

Here we cause the two numbers to be subtracted, instead of added, by turning the first into its negative equivalent (in 2's complement binary notation) before the second is added. This leaves their difference in the AC; but as we have no way of knowing which of the two numbers will be the larger, this difference may be either positive or negative.

As we wish to store the difference in positive form, it is therefore necessary to make the program perform a logical decision (diamond shaped box), and branch in one of two directions depending upon the result of that decision. If it finds the result is positive, it should store it away unchanged; but if the result is negative, it should convert it into the equivalent positive number before storing it.

This is done by forming the two's complement of the difference, in the same way used to negate the first number.

A program written in assembly language to perform this task might look rather like this:

```
START, CLA      /clears AC
      TAD NUMA  /adds first number
      CMA, IAC  /forms 2's
              complement
      TAD NUMB  /produces difference
      SMA      /is diff negative?
      JMP STORE /no, go to store it
      CMA, IAC  /yes, form 2's
              complement
STORE, DCA DIFF /store it
      HLT      /halt
NUMA,          /first number
NUMB,          /second number
DIFF,          /difference stored
              here
```

Here the labels "NUMA", "NUMB" and "DIFF" have been used for the three data storage locations, just to show you another possibility. Similarly a label "START" has been attached to the first instruction, mainly to clarify exactly where the program starts. This can be very worthwhile if there are data storage locations ahead of the instructions. The second last instruction has also been labelled "STORE", but for a more important reason which should emerge in a moment.

The conversion of the first number into its 2's complement is performed by a single instruction, the combined operate microinstruction CMA, IAC. The second number is then added to the AC as before, in this case forming the difference.

The logical decision and program branching is achieved by using the operate microinstruction SMA, together with a JMP instruction. If the difference stored in the AC is negative, the effect of the SMA microinstruction is to cause the program to skip the next consecutive instruction, so that it automatically goes to the seventh instruction (CMA, IAC) and forms the 2's complement of the difference as required, before storing it. But on the other hand, if the difference is already positive, the SMA microinstruction will not cause a skip, and the program will instead go to the next consecutive instruction.

This instruction is the JMP instruction, and its purpose is to allow the program to proceed directly to store the difference. This is achieved by giving the "DCA DIFF" store instruction the label "STORE", and writing the jump instruction as JMP STORE. If the program is coded by an assembler program, this will cause the correct coding to be automatically generated, taking into account the actual memory location of the instruction which it effectively is the operand of the jump instruction.

If it were to be stored in the computer's memory starting at location 000, the octal coding for this second program example would be:

Location	Code
000	710
001	111
002	705
003	112
004	724
005	070
006	705
007	313
010	721
011	(first number)
012	(second number)
013	(difference stored here)

This is a very simple example of a program involving a logical decision and so-called "conditional branching", but even so it illustrates that a program is not necessarily confined to

the simple execution of a linear sequence of instructions. The ability of the machine to test for certain conditions, and branch in various directions according to the result of the test, expands the whole scope and flexibility of programming considerably.

Of the operate microinstructions in EDUC-8's repertoire, there are three which are used in this way for conditional branching: SMA, SZA and their combination SZA, SMA. Instructions which may also be used for conditional branching are the IOT skip instructions SKF and SDF, and the memory reference instruction ISZ.

The ISZ instruction is in fact very powerful. One of its other applications is for another important programming technique known as "looping". The best way to illustrate this is with another simple example.

Let us say you want to repeat a certain sequence of instructions five times. One way to do this would be to simply repeat the group of instructions five times, so that the machine performed them one after the other. However this would tend to take up a considerable amount of memory. This space can be almost completely saved by arranging for the program to loop around and perform the single group of instructions five times, before continuing.

The principle used to achieve looping is shown in Fig. 3. Ahead of the sequence of instructions to be performed a number of times, a storage location is loaded with a number known as the "index", whose value is made equal to the 2's complement of the number of times the instruction sequence is to be performed. In this case the index is set to minus 5, as we wish to perform the sequence five times. This preliminary operation is known as "initialising".

After the actual sequence of instructions to be repeated, the index number is arranged to be incremented. Then the program is arranged to test whether the value of the index has become zero or not. If not, the machine is made

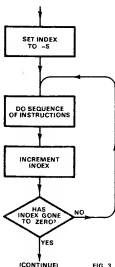


FIG 3

to jump back to the start of the sequence. If on the other hand the index has become zero, the machine is allowed to continue with the next consecutive instruction.

Because the index has initially been set to the 2's complement of the number of times the sequence should be repeated, it will not actually reach zero until the correct number of loops has occurred. Thus with our example where the index is set to minus 5, the program will automatically loop five times before proceeding.

The ISZ instruction can be used to perform

both the index incrementing, and the testing to see if its value has reached zero. The looping back is then arranged simply by following the ISZ instruction with a JMP back to the start of the loop, as follows:

```

      ...
      CLA
      TAD CONST
      DCA INDX  /initialises index to
              -5
GO,   ...      /sequence
      ...      /of
              instructions
      ISZ INDX  /increment index,
              test
      JMP GO    /not zero yet, keep
              going
      ...
```

Note that CONST would be a storage location with the number minus 5 as its content, while INDX would be another location used to store the index. The rows of dots represent the instructions before, inside and after the loop. Note that the loop may include any desired number of instructions.

Looping is a particularly useful technique, as you can imagine. It allows considerable reduction in the length of programs, as most programs involve a certain amount of repetition. Or looking at it in another way, looping allows more efficient use of computer memory, so that more elaborate programs can be fitted into the available space.

Note that the ISZ instruction is not the only one which can be used as the basis of looping. Broadly speaking, any of the skip instructions can be used to achieve looping, just as they can be used to achieve branching. The most appropriate instruction to use for looping depends upon the type of loop required—whether the program should loop for a fixed number of times (unconditionally), as in our example, or until some condition has been reached. The ISZ instruction is often the most suitable for the former, but instructions like SMA, SZA, SKF and SDF are generally more suitable for the latter.

An interesting example of the use of looping is where two numbers are to be multiplied together. Here one number may be used to control the number of times the other is added to itself, by converting the first number into its 2's complement and using it as the index for a loop around an instruction added the second number to the AC:

```

MULT, TAD NUM1
      CMA, IAC  /initialises
      DCA INDX
ADD,   TAD NUM2
      ISZ INDX  /finished yet?
      JMP ADD  /no, keep going
      HLT      /yes, halt
NUM1,          /first number
NUM2,          /second number
INDX, 000      /index stored here
```

Note that in this case the program would halt with the product of the two numbers in the AC register. It could be arranged to store the result in a suitable location, like the earlier examples, simply by adding the appropriate DCA instruction between "JMP ADD" and "HLT".

A flow chart for the example just given is shown in Fig. 4, to allow you to follow through the idea a little more easily.

Apart from its use in arranging program looping, the ISZ instruction can also be used to actually modify the instructions within the loop, so that the program need not simply repeat a sequence exactly the required number

## EDUC-8 computer

of times, but can perform a series of similar operations. An example of this is well worth looking at, because it demonstrates that instructions are not necessarily "sacred" and unalterable.

Say you have a set of eight numbers stored in consecutive memory locations, and you wish to add them all together to form their sum in the AC register. One way of doing this would be to have a sequence of eight different TAD instructions, each one referring to one of the eight locations. But it is more efficient to use looping around a single TAD instruction, and simply modify the address part of the instruction each time.

Thus if the locations in which the numbers are stored are given the labels "A", "A+1", "A+2", etc, the program would look like this:

```
START, CLA
      TAD CONST
      DCA INDX /initialises index
GO,   TAD A
      ISZ GO /modify instruction
      ISZ INDX /done B times?
      JMP GO /no, keep going
      HLT /yes, halt
CONST, 370 /minus B
INDX, 000 /first number
A,
A+1,
A+2,
...
```

Only the first three data number locations have been shown, we don't suggest the remaining locations. Note that the value given for CONST is the 2's complement of decimal B, in "377" format octal notation.

Of the two ISZ instructions used in this example, only the second is used "fully"—i.e., to both increment and test whether the operand has reached zero. The first ISZ instruction is used purely to increment the TAD instruction, and this is a perfectly valid way of using an ISZ instruction.

The only thing the programmer must do when using the ISZ instruction in this way is make sure that the number or instruction being incremented will never reach zero, or alternatively ensure that the program does not make an error if it does. In this example we know that the TAD instruction will only be incremented B times, so that there is no problem.

In cases where it is not easy to predict if zero will or will not be reached, and no branching or looping must occur, the best idea is to follow the ISZ instruction with a NOP or "no operation" instruction.

Although the method just illustrated can often be used to perform a series of similar operations, it is not always convenient to directly modify an instruction or sequence of instructions. For one thing, this complicates matters if the program is to be re-run, because the modified instruction or sequence of instructions will have to be changed back to their initial coding.

In such cases, use can be made of a similar technique, but one which uses indirect addressing via a "pointer". Here the pointer is incremented to change the memory locations referenced, not the basic instructions.

For an example of this, consider a situation where we have a set of eight numbers in consecutive memory locations, as before, but this time we want to add a constant—say octal 60—to each one, and replace it in its initial

location. A program to do this could take the following form:

```
START, CLA
      TAD CONST
      DCA INDX /initialises index
      TAD BUFSA /fetches add. of first no.
GO,   DCA POINTER /initialises pointer
      TAD 1 /fetch number
      TAD CON60 /adds 60
      DCA I POINTER /replace number
      ISZ INDX /increment pointer
      ISZ INDX /finished yet?
      JMP GO /no, continue
      HLT /yes, halt
CONST, 370
INDX, 000
BUFSA, /address of first no.
POINTER, 000 /pointer stored here
CON60, 060 /first number
A,
A+1,
A+2,
...
```

It is possible to use the one number as both index and pointer, in some situations, and this can be used to simplify and shorten the program still further. In the example just given, this could be done fairly easily, by arranging for the eight data numbers to be stored in the last eight locations in memory. The value of INDX each time would then automatically correspond to the required pointer address. You may care to try re-writing the program to do this, just for the exercise.

Along with the technique of using a pointer, the example just given also shows one important application of indirect memory addressing. Another important application of this should become apparent in a moment, as we discuss

another very useful programming technique—the subroutine.

Like looping, subroutines are used to allow a particular sequence of instructions to be used repeatedly during a program. But the advantage of the subroutine is that the sequence may be used in many different parts of a program, not necessarily in contiguous fashion. The sequence is made a separate entity, with the program jumping to it as required—and then returning back to where it jumped from, each time.

The general idea is shown in Fig. 5. As you can see, the subroutine is in effect a separate program module, to which the machine jumps from the main program whenever the subroutine sequence is needed. After having used the subroutine, the machine then jumps back to the main program, to continue on from where it was.

The instruction which is used to provide subroutines is the JMS instruction. This is a little like the JMP instruction, in that it has the effect of replacing the existing content of the PC register so that the machine takes its next instruction from a place other than the next consecutive location. But in contrast with the JMP instruction, where the existing content of the PC register is lost, with the JMS instruction the PC content is stored in the address given in the instruction, and the next instruction taken not from that address, but from the one after that.

This is used in the following basic way. The first location of a subroutine sequence is set aside as a storage location, immediately before the first of the actual subroutine instructions. Then whenever the subroutine is needed in the main program, a JMS instruction is used, specifying the address of the storage location. The effect of the JMS instruction is to store in that location the address of what would otherwise have been the next consecutive instruction in the main program, and to take the first subroutine instruction as its next actual instruction.

In other words, the JMS instruction causes the machine to jump to the subroutine, but at the same time stores the address in the main program to which it should return, in the subroutine storage location. This stored address is known as the "return address", for fairly obvious reasons.

To arrange for the machine to jump back into the main program at the end of the subroutine, it is merely necessary to make the last subroutine instruction an indirect JMP instruction, specifying the storage location at the start of the subroutine. The machine then automatically fetches the return address from the storage location, and places it into the PC register.

In terms of actual instructions, a subroutine and its associated JMS instructions tend to look as follows, when encountered in an assembly language program:

```
...
JMS SUBRT
...
JMS SUBRT
...
JMS SUBRT
SUBRT, 000 /stores return add. here
... /first sub. instructions
JMP I SUBRT /fetches return add. to exit
```

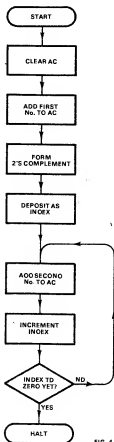


FIG. 4

Here the top section represents the

## EDUC-8 computer

main program, with JMS instructions wherever the subroutine is needed. The subroutine itself is beneath it, with its storage location for the return address at the start, and the indirect JMP instruction used to exit at the end. Note that although the label "SUBRT" has been used here for simplicity, any other suitable label may be used.

This is the basic way of providing a subroutine, and is probably the way most often used. It is possible to elaborate on the basic idea, for example if you want to transfer additional data numbers into and out of the subroutine (normally the only data transferred is the content of the AC register), but space limitations prevent us from going into this further here.

An important application of subroutines is in servicing input/output devices, and we will look at this shortly. However before we do, it may be worthwhile to mention briefly the general use of indirect addressing for memory reference instructions, in connection with memory pagination.

As mentioned in the earlier section dealing with basic machine operation, the eight-bit instruction words used in EDUC-8 do not allow direct addressing of all memory locations. This is because a full eight bits would be needed to address 256 locations, and at least three of the eight instruction bits are already required, for the operation code.

In fact, there are only four bits available for the actual operand address portion of a memory reference instruction, as bit 4 is used to indicate the addressing mode. As there are only 16 combinations possible with four bits, this means that any given memory reference instruction can only specify one of 16 partial addresses.

This problem is not unique to EDUC-8, but is in fact shared with many microcomputers. The only difference is one of scale—with a typical commercial machine, some 256 partial addresses may be specified, rather than just 16. And with EDUC-8, we get around the problem in the same way used in a commercial machine.

The convention adopted in the machine is to divide the memory effectively into "pages", in this case of 16 locations each. The full 256-word memory of the machine is thus considered to consist of 16 pages, each comprising 16 locations, and with octal addresses as shown:

000-017	200-217
020-037	220-237
040-057	240-257
060-077	260-277
100-117	300-317
120-137	320-337
140-157	340-357
160-177	360-377

In effect, the instruction decoding logic of the machine "assumes" that the partial address given in a memory reference instruction refers to the corresponding actual address lying in the same page as the instruction itself. Thus if a TAD instruction is in the first page of memory and has the octal code 105, the machine will regard the location specified as that with the address 005. However a TAD instruction with the same coding, but in the last page of memory, will be taken as specifying the location with the octal address 365.

This applies to both direct and indirect addressing, in the sense that the address of the actual operand address specified in an instruction is also assumed to lie in the same page as the instruction. So that if an indirect DCA

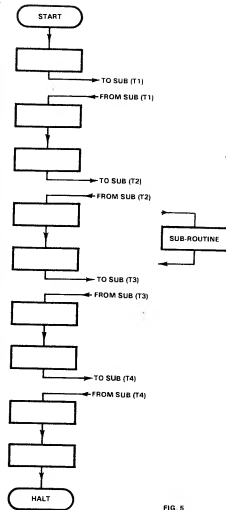


FIG. 5

instruction with octal code 313 is located in the first page of memory, in executing the instruction the machine will seek the number stored in memory at address 013 as the address at which to store the AC register contents. But if the same instruction code were encountered on the second last page of memory, the machine would seek the storage address at memory address 353.

From the programmer's point of view, the effect of all this is to restrict the range of locations which may be specified by a direct addressing memory reference instruction, to those locations within the same memory page as an instruction itself—known as the "current page". However if you want to specify a location outside the current page, all that is necessary is to use indirect addressing instead, with a location on the current page used to store the actual operand address.

An example should illustrate the idea. Say you are in the first page of memory, and need to deposit the AC contents into the location with octal address 246, in the eleventh page. The way to do so is to assign a suitable storage location on the current page, say at address 017, and store in that location the desired depositing address—octal 246. Then in the program itself all that is required is the indirect addressing DCA instruction with octal code 337.

If you find that "337" a little hard to work out, it is made up as follows. The first figure is the operation code for a DCA instruction, octal 3, which is no problem. The remaining

two figures are actually a combination of 017, the address of the current page storage location, and 020, which is the octal representation of the bit (bit 4) which indicates an indirect addressing memory reference instruction.

To round off this introductory look at programming, let us now consider briefly what is involved in programmed transfer of data to and from peripheral devices. This is usually called "I/O servicing".

As mentioned in an earlier section, I/O servicing involves three distinct operations. One is testing the flag line of the I/O device concerned, to see if the device is ready to "do business"; this is often described as testing flag status. The second operation is transferring the actual data, and the third is resetting the device flag. The latter not only prepares for the next servicing cycle, but generally also serves to indicate to the device that the data transfer is complete, at least from the computer's viewpoint.

The three instructions which perform these operations for an input device are nominally labelled SKF, KRS and RKF—although these mnemonics are arbitrary and could be varied to suit different types of input device. The second and third instructions may be combined to produce the instruction nominally called KRB, which transfers data and resets the device flag in the same execute cycle.

These instructions are used to service an input device such as the simple keyboard unit by arranging them in the following sequence:

TEST, SKF /is flag set?  
JMP TEST /no, keep looking  
KRB /yes, transfer and reset

As you can see, the SKF instruction is arranged to form part of a small program loop, by following it with a JMP instruction which forces the program to jump back to SKF again. While the device flag remains reset, the program thus "twiddles its thumbs" by jumping back and forth between the two instructions.

However as soon as the device flag is set, indicating that the keyboard has a character ready for transfer, the program can escape from the loop because the SKF instruction will cause the JMP instruction to be skipped. The machine will then fetch and execute the KRB instruction, transferring the character into the AC register and resetting the keyboard flag. Fig. 6 illustrates the technique in flow-chart form.

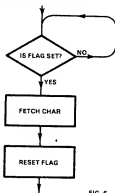


FIG. 6

A similar instruction sequence is used for servicing output devices, where the corresponding instructions are SDF, LDS and RDF, or LDB combining the second and third. However many output devices like paper tape punches and printers have drive mechanisms, which may be deactivated when the flag FF is set. For this reason, the instruction sequence

The program coding blank at right may help you in writing programs. Rather than use the blank itself, however, run off photocopies and use these.

for output device servicing are often rearranged thus:

LDB /load device buffer,  
reset flag  
TEST, SDF /device used char yet?  
JMP TEST /no, keep looking

By doing this the character is loaded into the device buffer first, and the flag reset to activate the punching or printing mechanism. Then the program is forced to wait until the device indicates that the character has been processed, before proceeding. As before this is achieved by using a JMP instruction to produce a small loop.

IOT device servicing sequences such as these are often needed many times in a program, and when this occurs it is usual to make them subroutines. Thus a program to make the machine act like an elementary desk calculator, using the simple input keyboard and octal display devices, might have two IOT servicing subroutines labelled "READ" and "DISPLY". Then whenever characters were to be read from the keyboard, this would be simply achieved by the instruction JMS READ, while to display a character would be achieved by the instruction JMS DISPLY.

The two subroutines themselves would be written as follows:

READ, 000  
TEST, SKF  
JMP TEST  
KRB  
JMP I READ

DISPLY, 000  
LDB  
BACK, SDF  
JMP BACK  
JMP I DISPLY

With EDUC-8, these are the only basic methods available for IOT device servicing. More complex machines generally offer the ability to service IOT devices in alternative ways, some of which may avoid the time-wasting flag test loop. A common approach is one employing an "interrupt" system, where the machine continues with its main program until a device signals its availability.

And with those brief details of IOT programming I must bring this section to a close. It has only been possible to deal with basic programming in a very limited way, and you will no doubt have many questions as yet unanswered. But hopefully you will now have enough basic insight into the concepts involved to provide a foundation for further study, as well as for trying your hand at some programs.

To help you in this regard, I have prepared artwork for a simple program coding sheet. This has space for 32 instructions, or two pages of memory, and provides for both writing the instructions in assembly language, and inserting the octal coding. As such, it can make program writing a little easier and more convenient. I suggest that you run the page through a copier, and provide yourself with a stack of copies to actually write programs on. This way you can keep the printed reproduction as a reference.

It has not been possible to give any examples of complete practical programs here, because of space limitations. However I will try to give some examples along these lines as we deal with further input and output devices. ©



## EDUC-8 PROGRAM

STEP	MNEMONIC	CODE
0		
1		
2		
3		
4		
5		
6		
7		
10		
11		
12		
13		
14		
15		
16		
17		
20		
21		
22		
23		
24		
25		
26		
27		
30		
31		
32		
33		
34		
35		
36		
37		



# Interfacing EDUC-8 with punched paper tape

Resuming the discussion of peripherals, the author explains here how to go about interfacing surplus paper tape readers and punches to your completed EDUC-8 microcomputer. Full logic circuits are given for typical reader and punch units, together with details of how to modify the circuits for other types. Software is also discussed, and a useful program presented to enable convenient punching of program tapes.

by JAMIESON ROWE

By this stage, quite possibly you have your EDUC-8 microcomputer and its two basic peripherals working, and have been trying your hand at some programs. If so, you've no doubt begun to find it tedious loading each program into the machine via the front panel switch register, and will probably be keen to provide a more convenient and speedy means of achieving the same result.

There are a number of alternatives to the manual method of feeding programs and data into a computer, as you are probably aware, and these generally involve a storage medium such as punched cards or tape, cards with optical marks or magnetic coating, magnetic tape, or a magnetic disc or drum. Each of these has its own combination of advantages and disadvantages, although some are now looked upon less favourably than others in commercial computer applications — where the emphasis tends to be on faster and faster operation.

For the present I am going to deal with only one of these storage media and its handling, although I hope to be able to discuss certain others in later instalments. The medium chosen for this initial foray into the subject is punched paper tape.

Why punched paper tape? Well, it seems to me one that is particularly well suited for storage of the modest-size programs used for an educational microcomputer like EDUC-8. The program tapes are short — at the most they are likely to be about a metre long, including leader and trailer. This makes them easy to handle, and quickly loaded into the machine even with a slow reader unit. At the same time the encoded information they contain is easily accessible, and can easily be "read" by eye after a little practice.

The equipment required to handle punched paper tape is also quite compact, and relatively simple.

There is also a very practical reason for the choice. I believe you are more likely to be able to pick up surplus punched paper tape equipment than any other. There is a lot of this equipment being "pensioned off" from commercial data processing facilities at present, it generally being replaced by key-to-disc systems.

Probably the most suitable sort of punch tape equipment to look out for in this context is the hardware making up what was generally known in the EDP industry as a "keypunch station". These were not connected directly to a computer, but were used merely for off-line preparation of data input tapes from type-writer-style keyboards.

There are two broad types of keypunch station. The simpler type was used for initial tape preparation, and has only a keyboard and tape punch unit together with the necessary power supply and encoding logic. The other type is known as a "verifier station", because it was used to compare the tape produced by the first operator with ostensibly the same information re-entered by a second operator (to try and detect, and correct, operator errors); this type of equipment has not only a keyboard and punch, but a tape reader as well.

The verifier station equipment is the one to aim for, in other words, because this will yield both a paper tape punch and reader, together with the necessary power supply unit to operate them. And for a bonus you'll get a keyboard unit — I hope to discuss how to use this also with the computer, later on.

If you're lucky, you may well be able to get hold of a complete verifier station for a few tens of dollars. Quite large numbers have been literally dumped in recent years, which seems almost criminal — particularly since most were made no more than twenty years ago, and cost more than a thousand dollars each!

By the way, it is important to make sure that you get tape equipment designed to take paper tape 2.54mm (1 inch) wide, and that it punches and reads the full 8 bit positions or "levels" possible with this tape. There are other types of equipment around, some of which use the same width tape but only 7 of the bit positions, while others use a narrower tape and either 5 or 6 bit positions. None of these would be

anything like as suitable for interfacing with EDUC-8 as is the 8-level tape equipment, because the instruction and data words for the machine are all of 8 bits.

To help you in checking that a set of equipment is designed for the correct 8-level tape, Fig. 1 shows the main features of this tape. It also shows the standard way of interpreting the bit position, with the three least significant bits on one side of the sprocket hole track, and the remaining five on the other.

Note that the type of encoding used in surplus keypunch gear is not important for our present purpose, as this part of the equipment is not used.

Incidentally, new punched tape equipment is available, if you can afford it. For example McMurdo Australia have a basic tape reader of Italian manufacture, the Ghielmetti type DTR 40K. This operates at up to 40 characters per second, but costs \$270.

The reader and punch unit pictured are both from a verifier station marketed by a big British computer firm, and are fairly typical. Both units were originally made by the Welmelec Corporation, of London. The reader unit is designated type RB.1, and the punch unit type S18.8. Both operate from a nominal 120V DC supply, and are capable of working at speeds up to around 10-15 characters per second.

The Welmelec RB.1 reader unit uses mechanical sensing of the punched holes, by means of eight small spring-loaded sensing pins. These are attached to a yoke assembly, which is moved upward towards the tape by a pair of heavy-duty solenoids. Each sensing pin is connected by a mechanical linkage to a set of switch contacts, so that the contacts close if their pin meets a hole and is able to travel its full upward stroke. Conversely if a sensing pin does not encounter a hole, it is prevented from moving its full stroke by the tape, and its contacts remain open.

At the top of its stroke, the yoke assembly trips a set of bi-stable switch contacts, which during the upward stroke have been closed. The contacts then latch in the open position, and because they control the current to the main actuator solenoids, this causes the yoke assembly to fall back to its rest position under the influence of gravity and a tension spring.

The yoke assembly has a pawl which engages with a ratchet mounted on the same shaft as the reader's tape sprocket. In falling back to its rest position the yoke assembly thus causes the tape to be incremented to its next row of punched information — after the sensing pins have been withdrawn. Finally, at the bottom of the return stroke, the yoke assembly trips the latching contacts of the bi-stable solenoid control switch, completing the cycle and readying the reader for another.

Note that with this type of reader mechanism, the number actually encoded on the tape is only "read" by the sensing pins, and reflected in their switch contacts, at the top of the stroke. This tends to make interfacing a little more complex than with other types, which generally

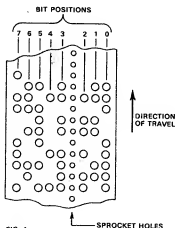


FIG. 1

Above are two views of the Welmelec tape punch, as modified by the author, with the power pack visible in the right-hand shot. At right is the companion reader unit, as modified for loading.

tend to read the tape in a static fashion—i.e., whenever the tape is not actually moving.

This means that the simple circuits I have developed for use with the Welmelec reader and similar "on the fly" types should work even more reliably with static types, and in that sense they should be suitable for almost any mechanism. On the other hand a static reading mechanism will generally work quite well with even simpler circuits, should you wish to take advantage of this.

Let us now look at the logic circuits required to interface a reader mechanism with our EDUC-8 computer. There are actually two different types of circuit, because there are two distinct functions which a reader may be called upon to perform.

One of the functions is to read input data into the machine, under the control of a program, in the same way data is fed in via the keyboard. This can be the major function of a reader with a large machine, but with a small machine like EDUC-8 it is likely to be of rather less importance than the second function: loading programs.

It is of course possible to load programs in the same way as data is read into the machine, under program control. Here the machine must contain a small program known as a "loader" or "bootstrap", whose sole purpose is to direct the machine in loading in the real programs. The bootstrap program may either be stored in the normal memory of the machine, after having been fed in manually via the console, or alternatively it may be provided by the computer manufacturer in "hard wired" form—i.e., part of the actual wiring, usually an auxiliary reader-only memory or ROM.

While a bootstrap program may be used with EDUC-8, this is not a very attractive proposition because the bootstrap would take up valuable memory locations. Even if it took up only 10 or 15 memory locations, this would be a significant and irking proportion of the total 256 available.

Happily this can be avoided, by providing a reader unit itself with the necessary logic "know how" to enable it to load programs automatically. Provision was made in the basic EDUC-8 design to allow this to be done, by means of the external "load address" and "deposit" signals.

Logic circuits will be presented here for both types of tape reader function. They will be presented separately, but could be combined if you so desire. The first, that for program loading, is shown in Fig. 2.

This circuit has been designed to load program tape punched in the following way. First, the tapes have a length of "leader", blank except for the sprocket holes; this allows the tape to be inserted into the reader gate without concern for the actual information. Then at the end of the leader, the start of the information itself is indicated by a single hole punched in the most significant bit position (track 7 in Fig. 1).

The row (or "frame", as they are called) immediately following the start bit is then used to contain the address of the first memory location in which the program is to be stored. After this, the succeeding frames contain the program itself, i.e., the actual instructions in the order they are stored in memory. No further address information is required after giving the first storage address, as the instructions are automatically stored in consecutive locations. The tape is arranged to end a few sprocket holes after the last instruction.

This format is illustrated in Fig. 3. Note that it is usual to cut the start and end of tapes as shown, to make it clear which end is which.

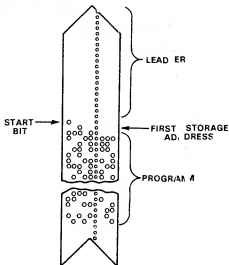


FIG. 3

The heart of the loader logic in Fig. 2 is the two RS flip-flops FFA and FFB, formed by the cross-coupled gates G2-G3 and G4-G5 respectively. These effectively form the "memory" of the loader, and control its operating sequence.

## EDUC-8 computer

When power is first applied, both flip-flops are reset by the R-C circuit at connected by diodes to the free inputs of G13 and G5. To initiate loading after a tape has been inserted into the reader, the operator merely presses the "load button". This takes the associated input of gate G1 to logic high, and because the other input of G1 will normally be so high (more about this soon), the output of G1 goes low.

The effect of this is to switch FFA to the set state, with the output of G2 high and that of G3 low. These current flows in transistor T2, via the series diode and resistor connected to the output of G3, and the gate of the C106B SCR, which triggers into conduction. The circuit is thus completed for the reader activator solenoids, and the reader mechanism thus starts operation.

Pulsed by the action of the latching reader switch contacts, which alternatively make and break the solenoid circuit at the bottom and top of the strokes, the reader begins scanning the tape.

A small low-voltage relay RLY A is connected via a 2.7k/10W dropping resistor in parallel with the reader solenoid circuit, such that its current is also alternatively switched by the latching reader switch contacts. A set of changeover contacts on RLY A is then used to generate a feedback logic signal, whose bounce transients are suppressed by the RS flip-flop formed by gates G8 and G9. When the reader is operating, the output of gate G9 is thus a rectangular wave, whose high-to-low transition occurs at the top of the reader's stroke.

This transition is important, because it corresponds to the instant in the reading cycle when the sensing pin contact have been stabilised in their "read" positions. Hence the signal from G8 is used to trigger three cascade-connected monostables (one-shot) D1, D2 and D3. The first monostable D1 is used to generate a pulse about 60µs long, immediately following the transition, while D2 and D3 produce two smaller pulses (about 3µs) which follow sequentially.

It is the third pulse, from D3, which plays the first part in the loading sequence, via gate G7. The second input of this gate is driven by a PNP transistor, T3, whose base is connected to the reader sensing contacts associated with bit position 1.

During 1 the sensing of the paper tape reader where there are no information holes punched, all the reader sensing contacts remain open. However, upon the arrival of the "start" frame, the contacts for bit 7 close at the top of the stroke, so that T3 conducts and takes the second input of G7 high. When the pulse from monostable D1 arrives at G7, the gate output therefore goes low, and this causes FFB to be set.

The setting of FFB causes two things to happen. One is that the inputs of gate G16 are taken low, so that its output goes high; this enables gates G17-G24, to open the data paths between the eight sets of reader sensing contacts and the switch register circuits of the computer.

It is second thing that happens when FFB sets is that gate G6 now has both inputs taken high, so that its output goes low and that of G12 goes high. This enables gate G14, so that at the top of the reader stroke for the next or successive frame, the 60µs pulse from monostable D1 causes G14 to deliver a LOAD ADDRESS (L) pulse to the computer. Because

the reader sensing contacts will simultaneously be sensing the initial loading address on the tape, this causes the computer to load the address into the PC via the switch register circuit.

This is providing, of course, that all of the switch register switches are in the "up" or 1 position, so that gates G17-G24 in the loader can control the logic levels. This is also necessary for correct depositing, so that when the loader is operated, the SR switches must always be in the up position.

Immediately after the address is loaded, the short pulse from monostable D2 arrives at the input of gate G10. As the second input of this gate is now high, because FFB has been set, the output of G10 is thus driven low briefly. And this in turn resets FFA, so that now only FFB is left in the set state.

Despite the resetting of FFA the reader mechanism continues to operate, because of the second diode and resistor connected from the base of T2 to the output of G5. Gate G6 is disabled, as the output of G2 has now gone low, but G11 is now enabled, because the outputs of G3 and G4 are now both high. This causes the output of G11 to go low, and the output of G13 to go high, enabling gate G15.

As a result, at the top of the next consecutive reader stroke, the 60µs pulse from monostable D1 is now passed by G15, to deliver a DEPOSIT (D) pulse to the computer. And this occurs when the reader sensing contacts are registering the first instruction, so that the computer will deposit this instruction as required in the first desired memory location.

This last sequence of events will be repeated for succeeding frames of the tape, depositing each of the instructions in turn. The loading sequence will normally only come to an end when the tape end sensing switch of the reader mechanism registers that the tape has run out. The switch breaks the 120V supply to the solenoids, stopping the mechanism. At the same time it causes the "tape fault" lamp to light, and resets FFB of the loading logic, by turning on transistor T1 via the 120k resistor connected to the latter's base.

When FFB resets, the output of G16 goes low, disabling gates G17-G24 and thus freeing the switch register circuits for normal operation. At the same time G15 is disabled, so that both the LOAD ADDRESS (L) and DEPOSIT (D) lines are also freed. And as both FFA and FFB are reset, the loader circuit is back to its initial state, ready for loading another program when required.

As you can see, the operation is not particularly complicated, yet the circuit performs the required loading sequence simply and reliably. It contains only eight ICs, of which six are of the low-cost "garden variety": 7400 x 3, and 7401 x 3. The remaining two are 9602 devices, to provide the monostable elements. Together with the ICs there are three low cost transistors, two 2N3638 or similar, and one BC108 or similar; also a C106B or similar SCR, some diodes and a handful of minor components.

Note that because the reader solenoids are highly inductive, they tend to generate a high back-EMF when their current is switched. This is the reason for the R-C circuit across the 120V switch contacts, for the EM408 diode, and for the R-C network across the SCR. The latter is to reduce the rate of rise in anode voltage, which would otherwise cause the SCR to turn on spuriously — and to its detriment.

The 10k pull-up resistors connected to the outputs of gates G17-G24 are effectively in parallel with similar value resistors on the front panel board of the computer itself. I have found them necessary in order to ensure completely reliable operation of the loader with "worst

case": 7401 gates, whose output leakage in the high output state can be as high as 250µA.

The additional resistors would not be required if the existing resistors in the machine were reduced from 10k to 4.7k, but this is not easy once the machine is assembled.

The pull-up resistors shown on the outputs of G14 and G15 are for the same purpose. Again it is easier to fit additional resistors in the loader, rather than to change the existing resistors in the machine itself.

Note also — and this is very important — that for correct loader operation, the 0.1µF capacitor shown bypassing the external deposit line of the computer should be removed, and replaced by a unit of 0.01µF. This is necessary to prevent faulty operation due to distortion of the loader's DEPOSIT (D) pulses from G15. The easiest way to make the change is to clip the original capacitor from the front panel board, and to wire the new one in at the 16-pin loader socket on the machine's rear panel.

The exact physical form of the loader logic just described will depend upon the reader unit you obtain, and this is left to you. In my case all of the circuitry except the SCR, its R-C circuit and RLY A, with its dropping resistor were mounted on one of our "Multi-DIP" boards — the larger size, having space for eight ICs. The other parts just mentioned were mounted on a small square of Veroboard, and both boards mounted inside the original reader case beside the mechanism.

The loader connects to the computer via a multi-way cable, which brings 5V OC to the loader as well as making the signal connections. The cable ends with a 16-way McMurdo plug, mating with the loader socket on the computer rear panel.

High voltage for the loader is derived from the original keypunch power pack, which is a simple transformer and selenium bridge unit with a 1000µF reservoir electro and a rating of about 2A continuous. I used a "Bulgin" 3-way panel-mounting plug on the loader, mating with a cord-type socket on the cable from the power pack.

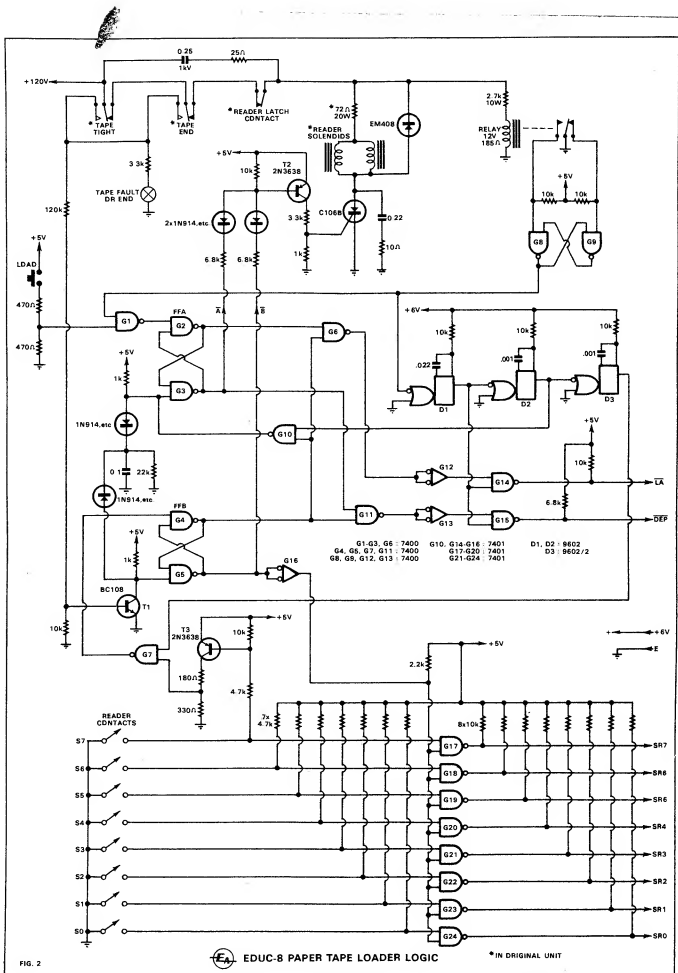
Note that the purpose of G1 in the loader logic circuit is to prevent the load button from setting FFA if the high voltage is not connected, or if the reader gate is open (which opens the tape end switch). When there is no high voltage, the contacts of RLY A revert to their normal positions, so that the output of G8 remains low and prevents G1 from responding to the load button.

This has been done to prevent the reader from suddenly "bursting into life" while the tape gate is being closed, if the load button has accidentally been pressed. Such a sudden start could well damage the tape, by moving it before the gate is properly closed and latched in position.

I should perhaps point out that the circuit of Fig. 2 does not show the 5V supply connections to the various ICs; these are assumed. Also not shown but nevertheless required are the usual bypass capacitors on the 5V line. I used two 0.1µF LV polyester capacitors in the loader, one where the 5V enters the board, and the other at the far end.

The other logic circuits given here will assume the same power supply and bypassing provisions.

As mentioned earlier, this loader circuit should be suitable not only for "on the fly" reader mechanisms like the Walmec R8.1, but also for those with static reading. However, if the reader mechanism uses a stepping motor or a motor and clutch escapement for tape drive,



### EDUC-8 computer

in place of the solenoids, you will need to modify the circuit slightly.

Such reader mechanisms are generally not capable of self-cycling like the Welmecc unit. They must usually be triggered into operation for each read cycle, by means of a pulse fed to the driver circuit controlling the stepping motor or clutch escapement solenoid.

As an example of what is required for this type of mechanism, Fig. 4 shows the changes I found necessary to adapt the loader logic for use with a type 1741 reader made by the Tally Corporation of Seattle. This reader has a synchronous motor and clutch escapement drive system, the clutch escapement being actuated by a small solenoid. This is driven with the appropriate length pulses (about 4, 5ms) by a power transistor fed by a monostable, both of which are built into the reader unit.

You can see from Fig. 4 that the main change to the logic is the addition of a simple clock oscillator, whose output takes the place of the feedback logic signal from G8 in Fig. 2. The gate formerly used for G8 is now used as an OR gate, which enables the clock oscillator when either FFA or FFB are set. The oscillator circuit is one that the author has used many times before — in fact the same circuit as used for the main clock generator in EDUC-8. It uses one half of a 7413 Schmitt trigger device, with a BC108 or similar NPN transistor.

The trigger pulses required for the monostable and driver circuits of the reader are derived from the output of D1, via the gate formerly

tively slowly to the drive pulse — compared with the 70-odd microseconds required for the logic to operate. In fact the tape typically does not start to move until about 3ms after the start of the drive pulse, and the sensing contacts do not begin moving for another 3 or 4ms. There is thus more than enough time to sense the tape before the sensing contacts are disturbed.

Note that there are other motor-clutch readers, like the Tally model 424, which do not provide the clutch solenoid driver or its monostable. For such readers the remaining 9602 element in the loader logic may be used to generate the required pulse (4.5ms long for the Tally 424), and a suitable power transistor used to drive the solenoid.

The Tally model 424 has either a 48V solenoid, of 220 ohms resistance, or a 24V solenoid with 50 ohms resistance. In either case a series R-C circuit of 0.25uF and 10 ohms should be fitted across the coil, and probably also an EM408 or similar diode as with the solenoids of Fig. 2. With these to suppress transients, the driver transistor could well be a 2N3055, or the smaller 2N3054.

The Tally readers may be fitted with a tape end sensing switch, but this is not connected into the clutch escapement circuit. It is left free, for use in the associated logic: I simply connected it as shown in Fig. 4, so that it merely turns on T1 to reset FFB as before.

Having looked at the logic required for a program loader, let us now turn to the logic required for program-controlled reading. This is considerably simpler, as you can see from Fig. 5. Note that this circuit has again been designed for a Welmecc type R8.1 reader, or

This produces a clean H-L transition at the output of G6, triggering monostable D1 and subsequently D2. In this case these produce pulses of length about 10 $\mu$ s and 1 $\mu$ s respectively.

The pulse from D1 is produced first, and this is fed to the parallel load enable inputs of the buffer register. This causes the information from the reader sensing contacts to be loaded into the buffer. Then D2 produces its pulse, and because the Q-bar output of D2 is connected to the free input of G2, this sets the flip FF.

When the flag FF sets, this sets the flag FF forward bias from transistor T1, preventing the stroke from conducting at the end of the return stroke, and hence stopping the reader. At the same time the flag line to the computer is driven low by G3, indicating to the machine that the reader has a character ready for transfer.

As soon as the program in the machine tests the flag line and finds the flag set, it can accordingly shift the contents of the reader buffer out, by feeding clock pulses in via G7. The number in the buffer leaves via G8 and G4, the latter being the line driver and the former an inverter to maintain correct logic polarity.

Note that when the contents of the buffer are shifted out, the buffer is automatically cleared because of the earthed PA, PB and DS inputs on the first 7496. This causes zeroes to be shifted into the buffer as the character is shifted out, clearing it ready for a new cycle.

The final phase of the reader cycle occurs when the computer sends its RESET FLAG (L) signal, which resets the flag and restarts the mechanism ready for a new cycle.

The RESET FLAG (L) signal is also fed to the L-bar inputs of the buffer ICs even though during normal reading the buffer will already be cleared when the signal arrives. The reason for the connection is that the buffer ICs will generally contain a spurious character when power is first applied, due to either the random turn-on effect or to the reader going through a spontaneous read cycle (before a tape is inserted). With the circuit as shown, the program can be arranged to pre-clear the buffer by an RKF instruction, before reading commences.

As with the loader logic, the circuit of Fig. 1 may be adapted for use with a stepping-motor motor-clutch type reader. The modifications for a Tally type 1741 reader are shown in Fig. 2. As before the main change is the addition of a clock oscillator which in this case is enabled directly by the output of G1 in the flag FF. The output of the clock oscillator connects directly to D1, as before.

Drive pulses for the clutch escapement monostable and driver are taken from the output D1, as before, in this case using the second 13 element as an isolating gate. Although the pulse is only 10 $\mu$ s in this case, shorter than the loader, this is still quite adequate.

If you are going to use the reader logic with fully type 424 reader, without the inbuilt monostable and driver, a second 9602 will be needed to generate the 4.5ms pulse, with a 3054 or 2N3055 as before.

Incidentally, the two 7496 devices used for buffer register in the reader logic could easily be replaced by a single device, one of the newer 74165 type. This is an 8-bit parallel serial register, more or less a mirror image of the 74164 in terms of function. The main logic change required if you want to use this device is that the parallel load pulse will need to be taken from the Q-bar output of D1, as an active low pulse is required.

The final logic circuit to be given is that for the punch, shown in Fig. 7. This has been

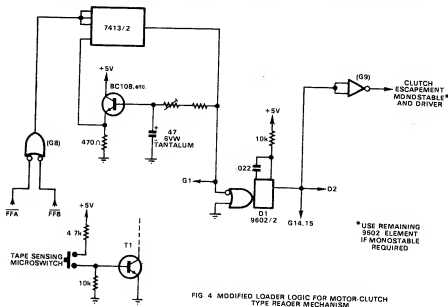


FIG. 4 MODIFIED LOADER LOGIC FOR MOTOR-CLUTCH TYPE READER MECHANISM

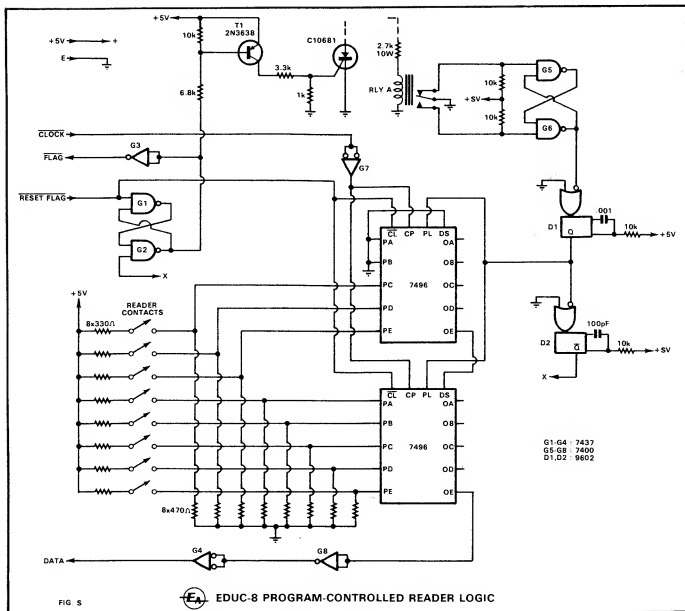
used as G9. The 60µs pulses from D1 are quite suitable for triggering the escapement monostable, so that the clutch increments the reader sprocket to move the tape one frame forward after each H-L transition in the clock oscillator output. The pot in the oscillator circuit thus becomes the loading rate adjustment, and with the Tally 1741 reader it can be set for speeds up to 50 characters per second.

As the clutch escapement pulse is being derived from D1, it might seem as if loader operation would be disrupted, with the tape being moved at the very time sensing takes place. However, this does not happen, because the reader clutch escapement responds rela-

a similar "on the fly" type.

Here there are only five ICs, of which two are 7496 devices which form the buffer register. The others are a 7437 forming the flag FF (G1-G2) and the line drivers, a 7400 forming the feedback bounce integrating FF (G5-G6) and the clock pulse and data inverters (G7 and G8), and a 9602 dual monostable for D1 and D2.

Operation is as follows. Initially, the flag FF formed by G1 and G2 is reset, with the output of G2 in the low state. This causes transistor T1 to conduct, triggering the SCR and starting the reader. Then at the top of the reader stroke, RLY A is turned off, along with the solenoids.

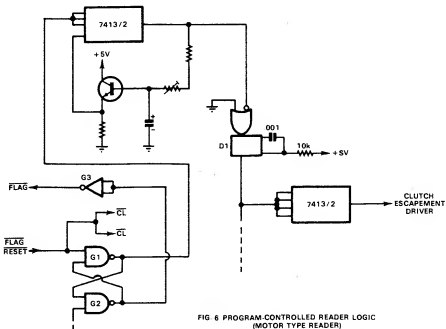


developed for the Welmec-type S18.8 punch, as pictured, although it should be fairly suitable for most surplus punch units.

Mechanically the Welmec punch is not unlike the companion reader, in that the drive is provided by a large solenoid and yoke system which can cycle itself via a bi-stable switch. However in this case the single drive solenoid operates not from the 120V DC supply, but from the 240V AC mains. Its current is therefore switched indirectly, via relay RLY B. A network of RF chokes and series R-C elements is used to suppress the very significant switching transient, with a sealed RFI filter in the mains cord for good measure.

When the yoke of the punch mechanism is pulled up by the solenoid, it is potentially able to force eight hardened punch pins through the tape, which is held between two die plates. But which of the punch pins are actually forced upward depends upon the position of each of eight spacer bars, controlled by the coding solenoids. Only those punch pins whose spacer bars are pulled into position by their coding solenoids actually punch holes, so the pattern of holes punched is controlled by the solenoids.

As you can see from the circuit, the punch logic is not very different from that for the





EDUC-8 PROGRAM		BINARY TAPES FROM KEYBOARD OCTAL CODE starts at 000	
STEP	MNEMONIC		CODE
0	RSTAD, 0		000
1	TEST, 017		011
2	JMP TEST		001
3	KRB		016
4	JMP 1 RSTAD		000
5	BUFF, 0		000
6	START, 017		016
7	JMS RSTAD //fetches A		400
10	DCA BUFF		305
11	TAD BUFF		105
12	LDB //display A		626
13	TAD BUFF		105
14	RAL		702
15	RAL		702
16	DCA BUFF		305
17	JMS RSTAD //fetch B		400
20	TAD 1 B'017		137
21	RAL		702
22	RAL		702
23	RAL		702
24	DCA 1 B'017		337
25	TAD 1 B'017		137
26	LDB //display AB		626
27	JMS 1 B'017 //fetch C		436
30	TAD 1 B'017		137
31	DCA 1 B'017		337
32	TAD 1 B'017		137
33	LDB //display ABC		626
34	JMP 1 C'017		535
35	CDIT, 017		100
36	RSTAD, 000		000
37	B'017, 000		000
40	JMS 1 RSTAD //fetch command		433
1	SA //punch?		724
2	JMP SCRB //no, scrub		310
3	CLA //yes, go ahead		710
4	TAD 1 B'017 //fetch word		134
5	LPB		636
6	CHNK, 017		631
7	JMP CHK		506
10	CLA		710
11	LDB //clears display		626
12	JMP 1 STAD //back to start		535
13	RSTAD, 000		000
14	B'017, 000		005
15	STAD, 000		007
16			
17			

You can use this simple program to punch out program tapes, using the simple input keyboard and display. It occupies only three lines of memory. Note that the constants identified with asterisks are initially in "377" format octal, but are then converted into "737" format to avoid possible confusion. The program starts at location 000.

to provide both 240V AC and 120V DC as required; the fifth wire is earth.

Apart from the power pack, the punch unit was made fully self-contained by transferring into its case the back space button and the "tape fault" lamp. These were originally in the small control console of the keypunch station. The old multi-way connector was removed from the side of the punch case, and replaced with a small aluminium plate. This mounts the back space and manual feed buttons, the tape fault lamp and the 6-way "Bulgin" captive plug used to connect with the power pack cable.

The connection to the computer is made via a length of multi-conductor TV remote control cable, as with the simple keyboard and

octal display units. The cable leaves the punch case via a grommeted hole, and terminates in a 6-pin DIN plug mating with the output device connectors on the rear panel of EDUC-8.

Since probably the main use of the punch with EDUC-8 will be to prepare program tapes, I have reproduced here a small program which allows you to punch program tapes easily using the simple keyboard unit and the octal display. In effect, the program is a very elementary "assembler", producing binary code from 3-digit octal numbers.

To use the program, connect the keyboard unit to EDUC-8's input device socket number 1, the octal display unit to output device socket number 0, and the tape punch to output device socket number 1. Then turn on both the computer and the punch power supply and, using the manual feed button on the punch, run off a suitable length of leader — say 20cm.

Now load the program into the machine, and enter 006 into the PC register as the starting address. Then set the machine running by pressing the run key. The program is now ready to begin punching a program tape.

First press the "4" key on the keyboard, and after this press the "0" key twice. The octal display should now show "400", and if it does, press the "LF" key. This will cause the punch to encode the start bit on the program tape, in bit position 7. The reason for pressing the keys to give octal 400 is simply that this corresponds to the binary number 10000000, in the "737" format assumed by the encoding program.

A similar process is used to encode the first leading address on the tape, and the actual instructions. Simply use the keys on the keyboard to enter in the octal code for the numbers to be punched, and if the octal display shows that you have made no mistake in keying in the three digits, press the LF key to have the resulting binary number punched.

If, on the other hand, the octal display shows that you have made an error keying in the digits, it is merely necessary to press one of the numeral keys instead of the LF key. This will cause the program to ignore the number keyed in, and clear its buffer ready for a new character. You can then try over again.

This is a very simple little encoding program, and you will no doubt be able to write more pretentious and elaborate ones yourself, if not immediately then later on. However, I have found it quite adequate for preparing program tapes and offer it to you as something to get you started.

Incidentally if you have a program already in the computer's memory, there is a quick way to punch it on tape for future use—providing that it doesn't take up all of memory. This is by using a "dump" program, written to punch out all of the numbers stored in a designated section of memory.

Such a program can also be used for duplicating tapes. If you have only a tape loader. Simply load the tape in, then dump it as many times as required. You may care to try writing such a dump program—it is a worthwhile exercise, and you'll find the program useful.

I hope the foregoing gives you a fairly good idea of the way in which punched paper tape equipment may be interfaced with your computer. As you can see, it is not difficult or unduly complex. If you are able to get hold of the necessary hardware, it is well worth the effort to make up at least the loader and punch units — not only for the exercise, but also because it makes program loading much more convenient.

A discussion of other peripheral devices will follow.

# QUAD

## 50 E

### PROFESSIONAL power amplifier



A single channel mono amplifier designed for Broadcast, Recording and other applications in the Audio Industry.

The multiple output windings of the Quad 50 terminate in a multiple output socket on the amplifier. Choice of matching is obtained by various connections in the output plug which is part of the installation itself. Quad 50 amplifiers may, therefore, be moved from one application to another without adjustment. With the exception of output impedance, the performance of the amplifier is identical with any output configuration.

The standard of quality is of the highest order and with any complex input, distortion falling within the useful part of the audio range will not exceed a small fraction of one percent. Overloading with any load will not significantly affect long time constants, ensuring immediate recovery and minimum distortion resulting from such overload.

#### Brief specification:—

**POWER RESPONSE**  
—1 dB at 30 Hz and 20 kHz ref to maximum output

**DISTORTION**  
40 Hz <0.35% any level  
1 kHz <0.1% up to  
10 kHz <1.0% minimum output

**OUTPUT SOLE IMPEDANCE**  
0.5 Ω in series with 25 μF for 5.5 A connection  
Others in direct proportion

**HUM & NOISE**  
Better than 80 dB referred to full output

**FREQUENCY RESPONSE**  
Unbalanced input  
—1 dB 30 Hz and 20 kHz ref: 1 kHz  
600 Ω bridging  
—20 dB 30 Hz and 20 kHz ref: 1 kHz

**INPUT LEVEL**  
0.5 V for full output, balanced or unbalanced  
Preset adjustment for higher levels

**INPUT IMPEDANCE**  
Unbalanced 4  
1 k—50 k Ω depending on preset gain  
600 Ω bridging  
<14 k Ω in parallel with 50 H

**STABILITY**  
Unconditionally stable with any load

Price and full particulars obtainable from the Australian Agent:

British Intertrading Pty. Ltd.,  
49-51 York Street, Sydney  
Telephone: 29-1871.



# Interfacing EDUC-8 with a Philips 60SR printer unit

Continuing the discussion of interfacing peripherals with the completed EDUC-8 microcomputer project, the author describes here the interfacing necessary for a Philips 60SR mosaic printer unit. With such a printer, your computer is able to print out messages, requests or the results of calculations in easily read form.

by JAMIESON ROWE

Most computers have a printing unit of some kind among their output peripherals, to allow them to communicate easily with a human operator and at the same time provide a permanent record of their interaction. With small machines there may be only the printing section of a teletypewriter, while larger machines tend to have one or more high-speed line printers as well.

Interfacing with a printer thus tends to be an integral part of most computer systems, so that the description of our EDUC-8 educational computer system would not really be complete if it did not include a discussion of this aspect. Even if you don't actually get around to obtaining a printer and hooking it up to your computer, I hope you will find the discussion of interest and value.

I have selected the Philips 60SR mosaic printer unit as the basis for discussion, for two main reasons. One is that it appears to be the only small printer unit offering a full alphanumeric character set which is readily available at the time of writing. The second reason is that Philips Elcoma very kindly made a sample unit available, to allow me to test and debug the power supply and interfacing circuitry!

The 60SR printer must be bought new, and as such will cost you considerably more than surplus paper tape gear. In fact, together with its associated electronics module, it will cost more than the EDUC-8 computer itself. This shows how far the cost of computers has fallen, thanks to modern IC technology, while the cost of mechanical peripherals like printers has tended to remain static. The cost differential is likely to increase even further in the future, unless we see some breakthrough in technology which will allow precision machinery to be made more cheaply.

In any event, the 60SR printer appears to be the most economical way of providing full alphanumeric printout facilities for a small computer like EDUC-8. The only possible exception would be a surplus teletypewriter, but these are in very scarce supply.

As shown in the photograph, the complete printing unit consists of the 60SR mechanism itself, together with two PC boards forming its associated CM64 electronics module. One PCB, known as the CC64 character circuit, provides the read-only memory (ROM) and scanning logic necessary to generate any one of 64 different alphanumeric characters, when commanded. The other PCB provides the power stages to drive the printer head solenoids from the CC64 output lines, it is known as the AC64 amplifier circuit.

The 60SR printing mechanism is a compact

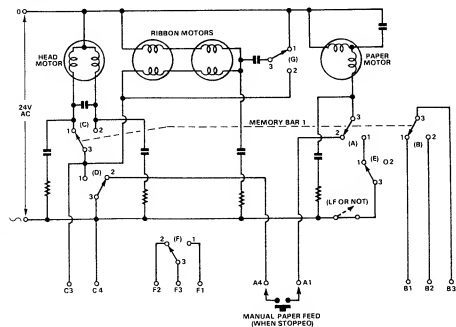
one, designed to print lines of up to 20 characters in length on paper 60mm wide. It takes readily available rolls of paper, as sold for adding machines and printing desk calculators. It takes standard typewriter ribbon, although an alternative version (60SA) is available which takes pressure-sensitive paper, and does not use a ribbon.

The character printing is not performed by the familiar raised-symbol hammers or "golf-ball", but by a set of seven blunt-tipped needles which strike the paper from the rear and force it against the inked ribbon. The needles are arranged in a vertical column, although the solenoids which drive them against the paper are disposed horizontally.

The needle and solenoid array form the printing head, which is driven horizontally across

HELLO, I'M EDUC-8 --  
SPEAKING TO YOU VIA  
THE PHILIPS 60SR  
MATRIX PRINTER

Above is a sample of the print-out, reproduced actual size. Fig. 1, below, shows the printer motor circuits.



the paper by a small synchronous motor. As the head travels across the paper, the needle solenoids are driven by the scanning electronics so that they produce the desired characters as a pattern of dots. Each character is formed as the head moves through approximately 1.5mm, corresponding to five needle-widths, and thus consists of a pattern of dots selected from a matrix or mosaic 7 dots high by 5 dots wide—hence the name "mosaic printer".

One advantage of this method of printing is that there are a very small number of moving parts compared with a conventional typewriter or teletypewriter mechanism, giving improved reliability. The fact that the characters are "formed" by the scanning electronics also makes such a printer very flexible, because the character set may be changed easily by exchanging the ROM in the electronics for another.

As a result of these advantages, mosaic printers are becoming increasingly used, and seem likely to replace other types eventually. The 60SR printer is therefore very much an example of "upcoming technology", as far as printers are concerned, although only a small one.

The printer mechanism uses four synchronous motors. One is for head drive, one for paper feed, and the remaining two for ribbon feed. The motors are controlled by a number of microswitches, connected as shown in Fig. 1. Note that switches C, A and B are operated by a bistable "memory bar", which is moved between its two stable positions by the printer head at each extreme of its travel. Similarly switch G, controlling the direction of rotation of the ribbon motors, is controlled by a second "memory bar" which senses ribbon tension.

*This is the complete Philips printer package 60SR printer mechanism, CC64 character generator board, and AC64 driver board. Only a small amount of additional logic is needed to drive them from the EDUC-8 computer.*

Switch D is used to sense whether or not the printing head is in its resting or "home" position, at the extreme left-hand side of the paper; it operates as soon as the head leaves this position. Switch F is arranged to operate when the head has moved a short distance from the home position, when the drive motor has reached full speed; it is used to signal the electronics module that printing may begin. Switch E is operated by the printing head near the right-hand extreme of its travel, and is used to time the operation of the paper feed motor.

The operation of the mechanism for a typical printing cycle is as follows. Initially the printing head is in the home position, and the switches are all in the positions shown. The cycle begins when terminals C3 and C4 are connected together externally, completing the head motor circuit through contacts C1-3.

The head motor starts turning, and moves the head away from the home position. Switch D then operates, closing contacts D1-3 and thereby latching the head motor circuit for the remainder of the cycle. This ensures that operation is independent of the external C3-C4 contacts, as soon as they have initiated the cycle.

After about 80 to 100 milliseconds the head motor has reached its full synchronous speed. The position of switch F is such that it is now operated by the head, and this can be used to indicate to the electronics module that printing can begin. As the head continues to move across the paper at constant speed, the solenoids then drive the printing needles against the paper under the control of the electronics module, printing the desired characters.

When the head nears the end of its forward travel, switch E is operated, opening contacts E1-3 and closing E2-3. This has no immediate effect, but serves to prevent the head motor from operating the instant that the head reaches the end of its travel and operates the main memory bar. When the bar is operated, it reverses switches C, A and B.

Switch contacts C1-3 are opened and contacts C2-3 closed, causing the head motor to reverse. The printing head thus begins moving back to its home position.

The reversal of switch A at the end of the forward head travel causes contacts A2-3 to open, and contacts A1-3 to close. However because contacts E3-1 are open, there is no immediate effect. It is only when the head has moved back towards its home position by a short distance, releasing switch E, that the paper motor is energised via contacts A1-3 and E1-3. The paper motor thus begins to feed the paper up, bringing the characters just printed into view and at the same time advancing the paper for the next line.

The paper motor only operates for a short time, however; just enough to advance the paper by about 5mm. It stops as soon as the head completes its return to the home position, operating the memory bar once again and reversing switch A.

Shortly before the head reaches the home position it operates switch F, forcing it back

to its resting position to prevent restarting of the electronics module until the head has regained full speed in a new cycle. And finally when the home position is reached, switch D is operated along with the memory bar switches, resetting all switches to their initial positions and de-energising the head motor until a new cycle is triggered by external closure of C3-C4.

During the whole of the cycle the ink ribbon motors have been energised along with the headmotor, via contacts D1-3. They have therefore moved the ribbon along as required, in a direction determined by the ribbon memory bar and switch G.

You will no doubt have noticed that switch B plays no direct part in the operation of the printer mechanism. Like switch F it is provided for control of the electronics module, primarily to inhibit printing during the return travel of the head.

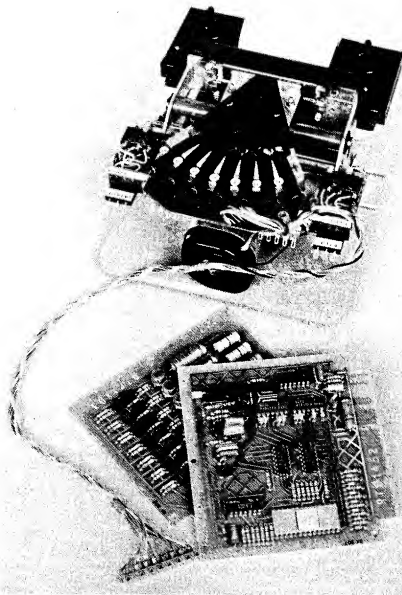
Note that the operation of the paper drive motor during the cycle may be inhibited, if desired, by opening the connection between switch contact E3 and its 24V supply rail. The paper may also be advanced manually, if desired, by using an external pushbutton or switch to join contacts A1 and A4 when the printer is stopped.

The CM64 character module provides all of the character generation and head driving electronics for mosaic printers like the 60SR. It has a repertoire of 64 different alphanumeric and punctuation symbols, which are generated in response to the application of a 6-bit character code number and a "start printing" (L) command pulse.

The 6-bit code used to specify which character is to be printed is a sub-set of the American Standard Code for Information Interchange, or "ASCII" code. A chart of the 64 characters available, together with the 2-digit octal equivalent of their 6-bit selection code numbers, is shown in Fig. 2. This table was actually printed by the 60SR printer itself, under the control of a small program running in the prototype EDUC-8 computer.

The heart of the CM64 module is the character generation ROM on the CC64 PC board. This is a 2240-bit device, organised as 64 35-bit words. Each of the 35-bit words is pre-programmed with the 5 x 7 mosaic dot pattern for a printing character. The 6-bit character selection number fed to the CM64 module is used as an address for the ROM, to determine which of the 35-bit words is made available to the scanning circuitry.

The scanning circuitry consists of a start cir-



## EDUC-8 computer

cuit, a scanning oscillator, an 8-bit shift counter and a decoder, and a set of gates controlling the ROM data outputs.

When the START PRINTING (L) input is applied to the module, it triggers the start circuit which enables the scanning oscillator. The shift counter begins counting, and its outputs are decoded by the decoder to sequentially select each of the 5 groups of 7 bits in the 35-bit ROM word corresponding to the vertical columns of the character to be printed. When the ROM data outputs stabilise each time, the output gates are enabled to allow the 7 bits

concerned to pass to the AC64 driver circuits, and thence to the head solenoids.

The dot information for each of the five vertical columns making up the character are thus read out and fed to the printing needles, one after the other, so that the character is printed as the printing head moves along the paper.

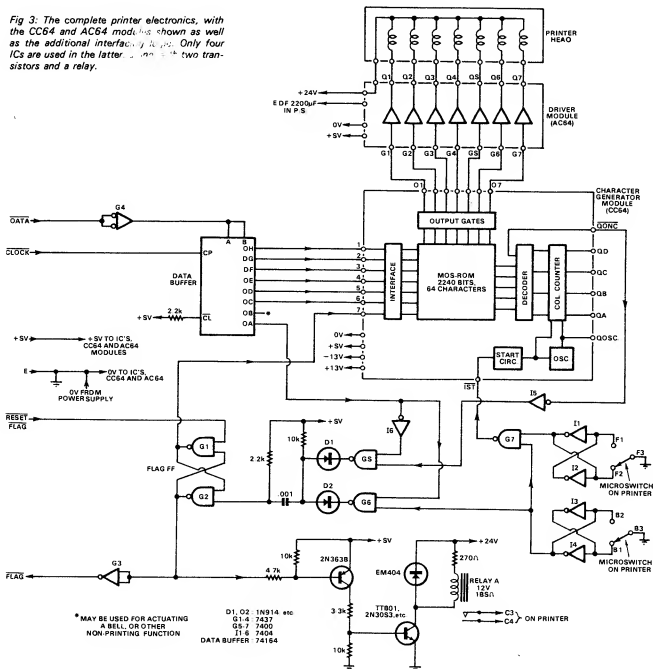
After the fifth column of data is read out from the ROM, the decoder generates a "Demand new character" (L) output pulse at the QDNC output, to signal that the current character has been printed and another should be supplied. At the same time the output gates are inhibited, so that the printing needles are prevented from operating. This continues for two scanning oscillator cycles, giving a space after each character equivalent to the width of two columns. The scanning oscillator is then turned

off, so that the scanning cycle for that character ends.

The scanning oscillator runs at approximately 400Hz, so that the columns of the characters are read out at approximately 2.5ms intervals. The exact scanning/printing rate can be adjusted, by means of a preset potentiometer on the CC64 PC board, to give between 18 and 20 characters per line.

Note, by the way, that the Philips 60SR printer is rather different from a teletypewriter, in that once the mechanism has begun to print a line, it must complete that line. This means that one cannot simply feed individual characters or words to the printer, at any convenient time. If you do so, it results in each character or word occupying a separate line, a rather wasteful and hard to read printout format!

Fig 3: The complete printer electronics, with the CC64 and AC64 modules shown as well as the additional interfacing logic. Only four ICs are used in the latter, plus two transistors and a relay.



EDUC-8 INTERFACING LOGIC FOR PHILIPS 60SR MATRIX PRINTER

To achieve the more usual and tidy format of having the characters and words grouped together to form reasonable length lines, the individual characters must be supplied to the CM64 module one after the other, as requested by the "Demand new character" signal. This means that before sending the first character of a line to the printer, you must make sure that the remaining characters for the line are available, to be sent after it at intervals of approximately 20ms.

The "whole line at a time" requirement of the 60SR printer does tend to make it a little unsuitable for certain applications, such as echoing an input keyboard. It also tends to complicate other applications such as printout of a text string, unless a "trick" is pulled in the interfacing logic. More about this shortly.

The AC64 amplifier circuit PCB of the printer electronics consists of the seven driver stages for the printer head needle solenoids. The driver stages turn on their appropriate solenoids, when selected, for the last half of each column scanning interval—about 1.25ms each time. The solenoids operate from 24V DC, each drawing approximately 850mA, when activated. The peak current from the 24V supply can therefore be as high as 6A during a 1.25ms printing interval (for example, when a full column of dots is being printed), but due to the 1:1 mark-space ratio and the variations between characters, the average peak current is nearer 1.5A.

The interfacing logic I have developed to connect the Philips 60SR up to the EDUC-8 computer is shown in Fig. 3. As you can see, it is fairly straightforward, and uses only 4 low-cost ICs, two transistors, a relay and a handful of minor parts.

You have probably identified by this stage the two main parts of the logic, because they are similar to those used in previously described peripherals: the flag FF, formed by gates G1 and G2, and the data buffer register, formed by the 74164 8-bit register.

The six least significant bit outputs of the 74164 data buffer register are connected to the six ROM address inputs of the CC64 character circuit, so that 6-bit words fed from the computer may be used to specify which of the 64 characters available is to be printed each time. The seventh "inhibit" input of the CC64 is connected to the flag FF, so that the ROM is not actually allowed to sense the address code until the computer resets the flag. This prevents spurious operation as the 6-bit data word is shifted into the register from the computer; by the time the flag is reset, the data word is fully settled in the buffer.

The two transistors form a driver circuit for the relay, which is used to activate the printer mechanism start contacts C3-C4. The input to the driver circuit is taken from the flag FF, so that the relay closes the circuit between the printer contacts whenever the flag FF is in the reset state. Thus when the computer resets the flag after feeding out a character code number, the printer mechanism is started at the same time as the ROM address inputs are enabled.

Note that the flag resetting does not itself trigger the start circuit of the CC64 character circuit. This is because the actual printing must be controlled by the printer mechanism. Hence if the character fed to the circuit from the computer is the first of a new line, printing will not begin for some time after the flag is reset—until the printer signals via switch F that the head is up to speed.

The START PRINTING (L) signal applied to the G1-bar input of the CC64 board is supplied by G7. This acts as an AND gate, fed by signals derived from printer microswitches F and B via

PHILIPS 60SR			
CHARACTER CODE			
(77 FORMAT OCTAL)			
CHAR	CODE	CHAR	CODE
0	00		40
A	01	!	41
B	02	*	42
C	03	£	43
D	04	%	44
E	05	2	45
F	06	£	46
G	07	/	47
H	10	(	50
I	11	)	51
J	12	*	52
K	13	+	53
L	14	,	54
M	15	-	55
N	16	.	56
O	17	/	57
P	20	0	60
Q	21	1	61
R	22	2	62
S	23	3	63
T	24	4	64
U	25	5	65
V	26	6	66
W	27	7	67
X	30	8	70
Y	31	9	71
Z	32	:	72
[	33	;	73
\	34	<	74
]	35	=	75
^	36	>	76
_	37	?	77

Fig 2: A listing of the characters in the CC64 character generator's repertoire, together with the corresponding octal/equivalent code numbers. As you can see, the listing was printed out by the 60SR printer itself, controlled by EDUC-8.

bounce suppression flip-flops. In contrast with the bounce suppression flip-flops in previous peripherals, these use inverter elements (I1-I2 and I3-I4), but work in the same way. Inverters are used here merely for economy.

Due to the action of G7, the CC64 start circuit is triggered into operation only for that part of the forward head travel in each printing cycle between the operation of switch F (signalling that the head is up to speed) and the reversal of switch B (signalling the end of forward head travel). It is prevented from being triggered during the entire return travel, and also for the first 100ms or so of the forward travel, before the head motor reaches full speed.

Gates G5 and G6, inverters I5 and I6 and diodes D1 and D2 are used to control which of two available signals are used to set the flag FF, to signal to the computer that a new character is required.

Predictably, the "demand new character"

signal from the CC64 board QDNC-bar output is one of the two signals, this being provided by the CC64 decoder precisely for such purposes. And this is in fact the signal normally used. It is fed to the input of G2 via I5, G5 and D1, and then via the differentiating circuit formed by the .001µF capacitor and the 2.2k resistor.

The differentiating circuit is necessary because the signal from the CC64 output is approximately 2.5ms long. If this were applied directly to the flag, it would block the flag in the set state for the equivalent of more than 25 computer instruction cycles. As a result the computer would keep sending new characters, in the mistaken belief that the printer was keeping up! The differentiating circuit prevents this from taking place by effectively converting the signal into a much shorter pulse—about 2µs—derived from the leading edge of the longer signal.

While setting the flag in this way using the "demand new character" signal is entirely adequate for many purposes, it is not sufficiently flexible to permit efficient and convenient printout of text strings. This is because of the "one full line at a time" requirement of the 60SR mechanism, which has no facility for executing a carriage return before the end of the line.

If the "demand new character" signal were the only signal available to set the flag, this would mean that each line of a text string would have to be ended with a number of spaces, designed to ensure that the printer did not reverse in the middle of a word. It would be tedious in the extreme to work out the number of spaces required for each line. Also, as the spaces must be stored in the computer's memory along with the rest of the text string, they would take up valuable memory space.

It is to avoid this that I have provided the alternative flag set signal. This is derived from the printer microswitch B, again via the bounce suppression flip-flop formed by I3 and I4, and is applied to the G2 input via G6, D2 and the differentiating circuit as before.

When the logic selects this flag set signal instead of the "demand new character" signal, the flag FF is not set until the printer head completes its current printing cycle, and returns fully to the "home" position. This means that the character whose code is currently applied to the ROM address inputs is simply repeated until the end of the current line. The printing electronics is then inhibited as usual during the return head travel, and the flag FF is reset only when the cycle is complete.

Thus if the code for space (octal 40) is applied to the ROM address inputs, the effect of selecting this alternative flag set signal is to automatically fill in the end of the line with spaces.

The selection of the two alternative flag signals is performed by the date word itself, fed into the buffer by the computer. This is possible because while the computer handles 8-bit words, only six of the eight bits are required by the CC64 ROM to specify the required printing character. The two remaining bits are thus available for auxiliary control functions, and I have used one of these to perform the required selection of set signals. The bit used is the most significant one, available at Oo of the buffer.

Thus if a character is to be printed normally, with other characters to follow in the current line, this is achieved by leaving the most significant bit a zero. This has the effect of inhibiting gate G6, but enabling gate G5 via I6. Hence the "demand new character" flag set signal is selected.

On the other hand if the character is required

## EDUC-8 computer

to be repeated until the end of the current line, all that is necessary is to make the most significant bit of its code number a one. This causes gate G5 to be inhibited, and gate G6 to be enabled instead, so that the "end of cycle" flag set signal is selected.

It may not be immediately apparent, but this facility greatly increases the efficiency and convenience of the printer. For example all that is necessary to end each line in a text string is to store a single "repeating space" (octal 240). This is simple, convenient and economical in terms of memory space.

In effect, a "repeating space" becomes the functional equivalent of a carriage return—line feed combination with other printers, and offers most of the flexibility this gives.

But this is not all, because the repeating facility may be used with any of the 64 characters in the 60SR's repertoire. And because you can make such a repeated character the first—and only—character of a line, this makes it possible to "stretch" a single character into a line the full printing width of the paper.

Thus for a line of asterisks all you need is a single character, with the octal code 252. Similarly code 255 gives a line of dashes (minus signs), or code 237 a line of underlines—either of which makes a very suitable way of providing top and bottom rules for list-out tables, etc. And the beauty is that they only require a single character code!

Note that the character repeating facility uses only one of the two "spare" data word bits, so that there is still one bit available for controlling any other function you may care to add to the printer. All that is necessary is to take the output from Ob of the 74164 data buffer, and provide the required circuitry.

One possibility would be to have this bit control the line feed facility of the printer, by using another small relay to perform the switching function shown dashed in Fig. 1. The relay could be driven from the Ob output of the data buffer via a transistor driver circuit rather like that shown for the printer start relay.

Another possibility would be to provide the printer with a "bell" facility, so that the computer can alert the operator when desired. Again this would involve a transistor driver circuit rather like that used for the printer start relay, in this case driving a small electric bell—or perhaps a Sonalert.

If you would like more than one such additional facility this could also be done, by using a one-of-four decoder driven by the two spare bits, rather than use the two bits direct. A suitable decoder would be one-half of a 9321 device, driven from the Ob and Ob outputs of the 74164. The existing flag reset logic could be driven from the "2" output of the decoder, leaving the other three outputs for the additional control functions. But note that as the outputs of the 9321 decoder are active low, the inverter I6 would need to be swapped over into the G6 gating input line, from the G5 gating input.

There are five distinct power supply requirements for the printer peripheral as a whole. The printer mechanism itself needs 24V AC, at a current of around 420mA. The printing needle solenoids and the relay run from 24V DC, with an average current requirement of around 1.5A (6A peak). The electronics in the CC64 circuits requires +13V DC at 30mA and -13V at 0.5mA, and +5V DC at about 370mA is required for the CC64, AC64 and the interfacing logic.

Of the five supply requirements, only the 5V DC supply is available from the computer, via the IOT connector cable. The remaining four must be supplied by a separate power supply built into the printer unit, and the circuit developed for this is shown in Fig. 4.

With a little ingenuity I have been able to provide all four of the required supplies from a single power transformer, one having two secondary windings each rated at 12V/1.33A. The transformer actually used is the Ferguson type PL30/40VA, one of their "low profile" 40VA family. This actually has two 15V windings, tapped at 12V, and it is the 12V portion of each winding that is used here.

The two windings are connected in series to form a centre-tapped 24V winding, which directly provides the 24VAC required for the 60SR printer mechanism. At the same time a bridge rectifier is connected across the full winding, and this is used both to produce both

floats at half the main bridge output voltage. A simple shunt regulator using a 13V/1W zener is used for regulation, with a 2200uF/16VW electrolytic across the output to reduce ripple to an acceptably low level.

Note that because the transformer winding floats at a DC voltage of approximately 18V above earth, neither side of the printer mechanism wiring may be earthed. This has no adverse effects, and no complications in terms of interconnections with the electronics—the contacts of the printer start relay simply float with the printer circuitry, while the contacts of micro-switches F and B connect only to the logic circuitry, and not to the printer circuit.

A separate half-wave shunt rectifier circuit connected to one side of the transformer winding is used to produce the negative supply voltage. Again a 13V/1W zener diode is used for regulation, and a 2200uF/16VW electrolytic for improved filtering.

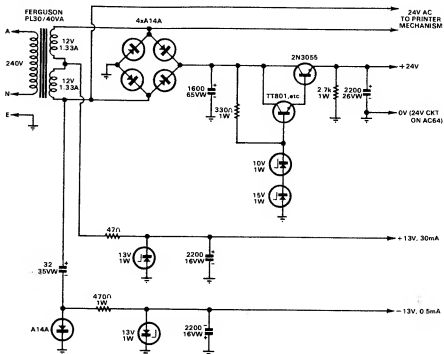


Fig. 4: The power supply circuit for the printer. This provides 24V AC for the printer motor, 24V DC for the relay and needle solenoids, and two 13V DC supplies for the CC64 character module.

the 24V solenoid-relay supply and the low-current +13V supply.

The full output from the rectifier bridge is fed to a 1600uF/65VW reservoir electrolytic. A simple series-pass regulator circuit is then used to regulate the output at the required 24V. The series-pass transistor is a readily available 2N3055 transistor, which is driven by a TT801, 2N3053 or similar TO-5 transistor of nominal 1W rating. Two 1W zener diodes, one a 10V type and the other a 15V are used to provide the reference. A 2000uF/25VW electrolytic is used across the output of the supply to provide the peak current capacity, with a 2.7k/1W bleed resistor to improve regulation.

Although the peak current drawn by the printer solenoids is 6A, this simple circuit maintains the 24V line well within the allowed tolerance at all times. In fact peak-to-peak ripple when printing a line of "B" characters (a fairly severe test) is barely 1V, and for typical lines is only about half this figure.

The +13V supply is derived from the centre-tap of the transformer winding, which

Although the total power consumption of all four supplies connected to the power transformer in the power supply can be as high as 50VA when the printer is actually printing a line of characters, the fact that printing only occurs for about 45% of the total printer cycle means that average load on the transformer is well within its 40VA rating. This is so even if the printer is used continuously, so that the transformer runs quite cool. In fact the main reason for using the 40VA transformer is to obtain satisfactory regulation.

Note that the five rectifier diodes used in the power supply are specified as A14 series devices. These devices are transient protected, and in view of their great ruggedness I recommend that you use them at least for the main rectifier bridge. Although the 100V rated A14A type is shown, the 50V A14V could be used instead if available, or failing these the more readily available A14D (400V). The price of all three types is very little higher than regular plastic diodes.

An important point to watch when connecting

## EDUC-8 computer

up the printer circuitry to the power supply is that the earth return of the 24V solenoid wiring should be run directly back from the AC64 PCB socket pin to the negative lug of the 2000uF electrolytic. There are heavy current pulses flowing in this lead during printing, and if they are not kept out of the rest of the electronics, the voltage spikes caused by lead inductance can cause all sorts of malfunction. The separate earth return ensures that this does not occur.

The remaining earth pin on the AC64 board should be run to a common earth point, established where the earth braid of the computer IOT cable is terminated. The earth connection of the CC64 board should also be made to this point, which is then linked to the earth pins of the various interfacing ICs, and back to the earthy lugs of the two 2200uF filter capacitors in the power supply. The 5V supply line from the computer should be bypassed to this earth line with .047uF LV polyester capacitors upon termination from the cable, at the AC64 and CC64 supply pins, and also near the 74164 and 7437 devices.

No details will be given here for the mechanical side of the printer, as this will depend upon the way you want to package it. As there are only four ICs, two transistors, a small crude relay and a handful of small parts in the interfacing logic, these can easily be mounted on a small PCB or square of Veroboard cut to match the CC64 and AC64 boards. The power supply wiring can be built up easily on a small length of miniature resistor panel, apart from the major parts.

Note that the CC64 and AC64 PC boards are designed to plug into 35-way edge connector sockets, having 0.1in connector spacing, and I suggest that you use sockets rather than solder direct to the boards, to avoid damage—they are not cheap. As the exact dimensions of sockets vary, it is best to obtain the matching Philips type F061 sockets (12-digit code 2422 048 13503); two will be needed.

Full connection details for the CC64 and AC64 boards is provided on leaflets supplied with them. Note that the CC64 is provided with connections from the scanning counter and the outputs from the oscillator, but these are not used here. The AC64 is also provided with duplicate output connections from the solenoid drivers—both as a row of pins of the top of the board, and as a group of pads on the edge connector. The latter are not used here, as the cable from the printer head is provided with a connector which mates with the row of pins.

The connections for the printer mechanism are clearly marked along the terminal board at the rear of the frame, and also on small connectors supplied "plugged on" to the various connector pins. The connection code is the same as used in Fig. 1, so that you should have no trouble in wiring this up.

Finally, a word about programming for the printer. Basically, the printer is handled in much the same manner as any other output device, the usual way being to service it via a subroutine which transfers the data word out to the device buffer from the AC register, resets the device flag, and then waits until the flag is set again.

The data words sent to the printer will be the codes for the particular characters to be printed, supplemented by the control bit—0 or 1, if you have wired the printer to respond to both.

If you have wired the printer according to

EDUC-8 PROGRAM			ADDRESS	DATA
STEP	ADDRESS	DATA	CODE	
0	START, C00		71H	
1	71H, BFF	fetch start of buffer	31H	
2	00A, PFF	initialise printer	21H	
3	00A, 71H	1st char	39H	
4	02A, PFF	terminate code	73H	
5	01P, +2	/no, keep going	59H	
6	01H, /yes, stop		72H	
7	01H		62H	
8	01H	print sequence	62H	
9	01P, -1		51H	
10	132, PFF	/inc-mark pointer	21H	
11	01P, 00H	/continue	59H	
12	01H, /end of buffer -- no test		72H	
13	01H		---	
14	01H		---	
15	01H		---	
16	01H		---	
17	01H		---	
18	01H		---	
19	01H		---	
20	01H		---	
21	01H		---	
22	01H		---	
23	01H		---	
24	01H		---	

Fig 5: A simple program to print out text strings, from a buffer starting at location 020. Note that the instructions in 005 and 011 use a full point to represent the "current memory location"; this is a fairly common mnemonic convention.

Fig. 3 the basic character code will be as shown in Fig. 2, with the control bits normally both being set to zero. To make any desired character repeat until the end of a line, the code is simply modified by setting the most significant bit to 1—equivalent to adding octal 200.

If a program must print out an instruction to the operator, an explanation or some other string of text characters, this is normally done by storing the appropriate string of code numbers in sequence in the memory, as part of the program. The part of memory occupied by the text string is usually called the "text buffer". To print the text the program is provided with a small instruction loop, wherein an indirect TAD instruction using a "pointer" address is used to fetch the code numbers from the buffer locations one after the other, by incrementing the pointer address each time around the loop.

As the "at" symbol corresponding to octal code 00 is very rarely used in text strings, this can be used as a message terminate code. Thus the string of characters forming the message in the text buffer need only be followed by a 000, and the instruction loop used for the printing out arranged to jump out of the loop immediately it recognises that the number fetched from the buffer is zero. This allows the message itself to time the number of times the loop is traversed, and avoids the need to set a loop counter to different values for different length messages.

The simple program shown in Fig. 5 should help to illustrate these techniques. It is purely a program to print out a "pre-recorded message", and you can make it print out any message you like simply by depositing the appropriate character codes in the locations beginning at that with address 020. Note that if you do not end the desired message with a 000 terminate code, the program will continue to print out whatever random numbers are present in the rest of the memory, and will only stop when it has printed the contents of location 377.

This illustrates the very simplest type of programming for a printer, where the characters are simply stored and "played back". There are many other possibilities. For example the program which was used to print out the table of Fig. 2 used this technique for the text at the top of the table, but printed the table itself by using an index variable which was printed out to give the first character, then analysed to produce the codes for the equivalent octal digits, which were then printed out also—after printing 6 spaces. Then the index was incremented, and the process repeated to produce the second line. This was arranged to be repeated for a total of 64 times (77 octal), to print out all 64 characters and their octal codes.

Space prevents me from giving the program itself here, and in any case it would be a good exercise for you to try writing one yourself, knowing what has to be done and the general line of attack. Here's a clue, though: to print out a 3-bit binary number as the equivalent octal digit, all you need to do is add octal 60, to get the required code number.

Incidentally, in writing programs which involve the printer, don't worry too much about the fact that it needs to be fed with a stream of characters once a line printing cycle has been started. Even though EDUC-8 is not a particularly fast computer, it is still quite fast compared with the printer.

In fact the time interval available to the computer to generate and deliver a new character, after the printer flag has been set by the "demand new character" signal, is typically 6.25ms. This is equivalent to about 65 fetch-execute instruction cycles—so that there should normally be more than enough time for the computer to keep up. Unless you have a very complex program, it is more likely to be spending most of its time waiting on the printer!

I hope the discussion given here of the Philips 60SR printer and its operation with the EDUC-8 microcomputer is of interest to you. While only a small printer, it makes a very worthwhile addition to the overall EDUC-8 system, and illustrates very well the way in which larger printers are used.

## PLEASE NOTE

Further testing of the interfacing logic given for the Welmelec paper tape punch, in the last section, has revealed that malfunction can occur during fast repetitive punching, due to a timing problem.

Because RLY A, used to generate the "end of cycle" flag setting signal, is driven from the top contact of the bistable switch contacts, it can reset the flag slightly before the cycle has actually ended. If the computer immediately loads in a new character and resets the flag (still before the cycle has actually ended), the punch will ignore the new character. At the same time the computer program continues to wait for the punch to set its flag, to indicate that it has punched the character. Thus the system "hangs up".

There are a number of ways in which this can be prevented, but the nearest and most effective way is to simply slow the operation of RLY A down, using an R-C circuit between the top contact of the bistable switch and the +120V supply line (in parallel with the existing spark-suppressor capacitor). I found that a 4.7k/150V electrolytic in series with a 220 ohm 1W resistor did the job very well.

# A full ASCII-type input keyboard for EDUC-8

Continuing the description of suitable peripheral devices to interface with his EDUC-8 microcomputer project, the author here gives details of a full alphanumeric input keyboard unit. The logic is capable of fully encoding all 128 characters of the standard 7-bit ASCII code.

by JAMIESON ROWE

Although the simple 16-key input keyboard unit described earlier is likely to be adequate for many purposes, there will no doubt be some constructors who will want to provide their EDUC-8 microcomputer with a full typewriter-style keyboard unit. Apart from offering increased communication flexibility and convenience, a full keyboard also allows you to work with programs requiring full alphanumeric input, such as symbolic editors, assemblers and compilers.

With this in mind I have developed a design for a full ASCII-type input keyboard, which will be described here.

The logic circuit has been designed to work with any keyboard assembly having the required number of switches, providing these can be used as SPST or "form A" switches (normally open, but closed when the keys is pressed). The switches need not be bounceless, as the logic will cope with any normal amount of contact bounce.

I have built up the design and checked it out with a surplus keyboard switch assembly, typical of the type available in obsolete keypunch equipment. This type of keyboard is likely to be that most accessible to you, at reasonable cost. Providing the switch contacts are in reasonable condition, such a keyboard should be quite suitable, although most will need a thorough dusting, lubrication and treatment with contact cleaning fluid before use.

You can, of course, elect to wire up the keyboard using a new keyswitch assembly, of which a number are available—at an appropriate price. For example General Electronic Services (99 Alexander St, Crows Nest, NSW

2065) can supply a 56-key keyboard assembly built by Mechanical Enterprises, Inc., of Virginia. Designated type AA-L2-R2, it has gold contacts, an 8-keylength space bar, and two 1½-keylength shift keys. The price is approximately \$130.

A new keyswitch assembly will undoubtedly give you a better looking keyboard, and ultimately greater reliability. However, as the cost of a surplus keyboard is unlikely to cost you more than about \$10-20, there is a considerable cost incentive in "making do" with one of these.

The keyboard unit shown in the pictures is from the same surplus keypunch station which provided the Welmeec reader and punch units previously described. It is fairly typical of the sort of keyboard switch assembly you are likely to obtain from such sources, solid in construction and still quite serviceable despite many years of work.

The keyboard logic presented here is capable of encoding all 128 characters of the standard 7-bit ASCII code, in contrast with some of the alphanumeric keyboard designs published elsewhere. In addition, the logic incorporates reverse shift mode encoding, for those keys where this is normally used.

Incidentally, if you've forgotten what the acronym "ASCII" stands for, it is the American Standard Code for Information Interchange.

For the benefit of those not too familiar with the ASCII code, it is shown in Table 1. As you can see, the 128 characters are arranged in eight columns, each column having 16 rows. The 16 row positions are defined by the four least significant bits of the code, bits 0-3, while the three remaining bits are used to define the

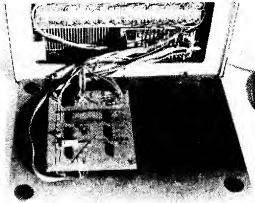
columns. Each one of the 128 character cells in the array is therefore defined by a unique combination of the 7 encoding bits.

Columns 2, 3, 4 and 5 comprise the printing characters normally used on teletypewriters and printers (such as the Philips 605R printer, for example). These columns include all of the upper-case alphabetic symbols, the decimal numerals, and all of the commonly used punctuation marks. The 64 characters concerned are often considered as a distinct sub-set of the full 128-character code, known as "6-bit ASCII".

As far as keyboards are concerned, the characters in columns 4 and 5 are normally provided with keys of their own, while those of columns 2 and 3 are usually arranged to share common keys, with a "shift" key used to change the encoding and distinguish between one column and the other. Usually the characters in column 3 are the "normal" characters for these keys, and those in column 2 the "shift" characters—except for the character pairs in rows 12, 13, 14 and 15. These usually follow the reverse convention, with comma, minus, fullstop and oblique the "normal" characters and their corresponding characters the "shift" equivalents.

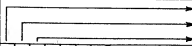
This special treatment of the four keys concerned tends to complicate the logic, as we will see shortly. However it has apparently become standard because of the greater use normally given to the four characters in column 2, compared with those in column 3. It would be a nuisance having to press the shift key every time one wanted a comma or fullstop, for example.

Columns 6 and 7 of the table comprise the lower case alphabet and some infrequently used symbols and control codes. Many keyboards do not provide for these codes to be generated, apart from "del" (delete or rubout) and perhaps "alt mode". This is because lower case characters are not necessary in most communication, and are not even available on many printers. For



At left is the prototype keyboard unit, which is built inside the case of a surplus keypunch keyboard. The view above shows the logic board, mounted on the bottom plate of the case.

TABLE 1: ASCII CHARACTER CODE

								TABLE 1: ASCII CHARACTER CODE							
								0	0	0	0	1	1	1	1
								0	0	1	1	0	0	1	1
								0	1	0	1	0	1	0	1
B 6	B 5	B 4	B 3	B 2	B 1	B 0	COLUMN →	0	1	2	3	4	5	6	7
							ROW ↓								
							0	NUL	DLE	SPACE	0	@	P	\	p
							1	SOH	DC1	!	1	A	Q	a	q
							2	STX	DC2	"	2	B	R	b	r
							3	ETX	DC3	# (2)	3	C	S	c	s
							4	EOT	DC4	\$	4	D	T	d	t
							5	ENQ	NAK	%	5	E	U	e	u
							6	ACK	SYN	&	6	F	V	f	v
							7	BELL	ETB	'	7	G	W	g	w
							8	BACK SPACE	CAN	(	8	H	X	h	x
							9	HOR. TAB	EM	)	9	I	Y	i	y
							10	LINE FEED	SUB	*	:	J	Z	j	z
							11	VERT. TAB	ESCAPE	+	;	K	[	k	{
							12	FORM FEED	FS	,	<	L	\	l	}
							13	CARRIAGE RETURN	GS	—	=	M	]	m	} (ALT MODE)
							14	SHIFT OUT	RS	.	>	N	^ (†)	n	
							15	SHIFT IN	US	/	?	O	_	o	DEL (RUB OUT)

Above is a table showing the 128 characters of the 7-bit ASCII code, with their coding. At right is the standard keyboard format used for this code.

the same reason it is usual for the upper case characters to be the "normal" characters, with the lower case characters in columns 6 and 7 encoded by using the shift key—just the opposite of the convention used with typewriters.

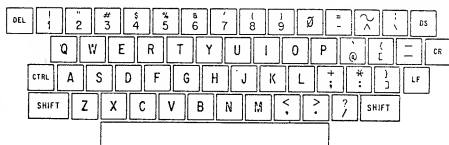
The characters in the remaining two columns, columns 0 and 1, are known as the non-printing or "control" characters. These are used to indicate changes in encoding mode, to control printer formatting, and for other facilities which may need to be controlled by "transparent" character codes—distinguishable from printing characters.

Generally most of the codes in columns 0 and 1 are generated using a "control" key, which acts rather like a second shift key. Any desired code in column 0 is generated by pressing the control key in conjunction with the key in column 4 corresponding to the desired row code. Thus "form feed" may be generated by pressing "control" and "L", while "horizontal tab" is equivalent to control-I.

In some cases special keys are provided for column 0 character codes, because they are used fairly frequently. This is generally done for "carriage return", "line feed", "back space" and "bell"—the last of these being used to activate the signalling bell of a teletypewriter.

The codes in column 1 are normally generated by using the control key in conjunction with the keys for columns 3/2, although some teletypewriters provide a special key for "escape".

Note that the characters in the ASCII code are not completely rigid, with certain codes



being used to represent two different characters on occasion. Thus the character in row 3 of column 2 is generally the "number" sign on American equipment, but the "pound" sign on machines of UK origin. Similarly the code in row 13 of column 7 is sometimes used to represent "wiggly closing bracket", and sometimes the non-printing control key "alt mode".

All of the character codes shown in the table may be generated using the logic circuit which will now be described. As many special keys as desired may be used to generate column 0 and 1 codes, depending upon the number of keys available on the keyboard you use. The remaining codes may be generated using the control key. Similarly the shift key may be used to generate the characters in columns 6 and 7, although "del" may be provided with a special key if you so desire.

As you can see from the circuit diagram, the main alphanumeric keyswitches are wired in an array, whose rows and columns correspond to those of columns 2, 3, 4 and 5 of Table 1. There are 16 "row" keylines, and three "column" keylines—two of which correspond directly to columns 4 and 5 of the table, while the last corresponds to both columns 2 and 3 (which use common keys).

At the heart of the encoding logic is an IC specially designed to perform keyboard encoding. This is the HD-0165, made by Harris Semiconductor in the USA. It is available on order through your usual parts supplier, from the local agents for Harris, Cema Distributors Pty Ltd of 21 Chandos Street, Crows Nest, NSW 2065. At the time of writing it costs about \$9, but performs a job which would otherwise involve many diodes, transistors and ICs.

The HD-0165 is a 16-line to 4-line binary encoder, which also generates both a "stroke" or "key pressed" signal and a "rollover" or "more than one key pressed" signal.

As you can see from the circuit, the HD-0165 is used to perform the basic encoding of the four least significant bits. As this corresponds to the row encoding for Table 1, its 16 inputs accordingly connect to the 16 "row" keylines of the keyswitch array.

Encoding of the remaining three bits of the ASCII code is performed by the logic circuitry involving gates G6, G7, G10-12, and inverters I1-6, using signals derived from the "column" keylines. The actual column keyline signals are generated by PNP transistors T1, T2 and T3.



# amazing MAGNA-LITE

lets you focus on  
fine detail regardless  
of lighting!

## magnifier flashlight

**Illuminates and magnifies  
whatever you are viewing!**

What a useful combination!  
A concentrated light beam and  
magnifier all in one! So compact  
it fits in your hand, pocket,  
purse or glove box.  
Precision made in the United  
States, amazing Magna-Lite  
is yours for just \$5.95!  
With a money back  
guarantee! Hurry while  
shipment lasts! Mail  
no-risk coupon today!



- Optical magnifying lens.
- 2 batteries included.
- Powerful flashlight.
- Lightweight, only 1 1/2 oz. fits pocket purse or desk.
- replacement batteries readily available at most chemists.

**MONEY  
BACK  
GUARANTEE**

**HOW TO ORDER:** Fill in BOTH sections of the coupon. In each section print your name and full postal address in block letters. Cut the entire coupon out around the dotted line. Send it with your money order or crossed cheque to Electronics Australia, Reader Service (Magna-Lite) P.O. Box 93, Beaconsfield, N.S.W., 2014. Cheques should be endorsed on the back with sender's name and address and made payable to Electronics Australia. This offer is open to readers in Australia only.

### CLIP COUPON — MAIL TODAY!

Name .....	
Address .....	
State .....	Postcode .....
Electronics Australia Magna-Lite Offer No. 36/75	
Name .....	
Address .....	
State .....	Postcode .....
Please send me ..... (state quantity) Magna-Lite/s. I enclose my cheque, postal/money order to the value of \$.....	

## EDUC-8 computer

These are wired so that when any of the keys in the associated keyline is pressed, the resulting input current drawn by the HD-0165 passes through the base-emitter junction, turning the transistor on.

Further logic signals are generated by the "shift" keyswitch, and the "control" keyswitch. Special keys such as "carriage return", "line feed" (LF), "back space" and "bell" are arranged to have the same effect as if the control key were pressed along with the corresponding column 4 alphabetic key, by means of diodes D3-D10. Thus D3 and D4 ensure that the carriage return key generates the "control" and "M" signals, as well as causing T3 to conduct.

As the "space bar" key is equivalent to "shift-0", diodes D1 and D2 are used to achieve a similar result when this key is pressed. Here the key causes activation of the "0" input of the HD-0165, generation of the "shift" signal, and conduction of T1—all three of which are needed to achieve the same result as shifted zero.

The actual encoding for bit 4 is generated at the commoned outputs of gate G6 and inverter I3. These are both open-collector elements, with their outputs tied together to achieve a wired-OR function. Similarly the bit 5 encoding is generated at the commoned outputs of G7-14, and the bit 6 encoding at the commoned outputs of I5-16.

The inputs of these elements are fed with the column keyline logic signals from T1, T2 and T3 together with the control and shift keyline signals, to achieve the correct encoding. For example I4 is fed with the control keyline signal, to ensure that bit 5 is false (0) whenever the control keyline is activated—corresponding to a column 0 or 1 character. Gate G7 is fed with the inverted shift keyline signal from I1, together with a wired-OR combination of the column 4 and 5 keyline signals from T3 and T2, so that bit 5 is again made false whenever a column 4 or 5 key is pressed without the shift key.

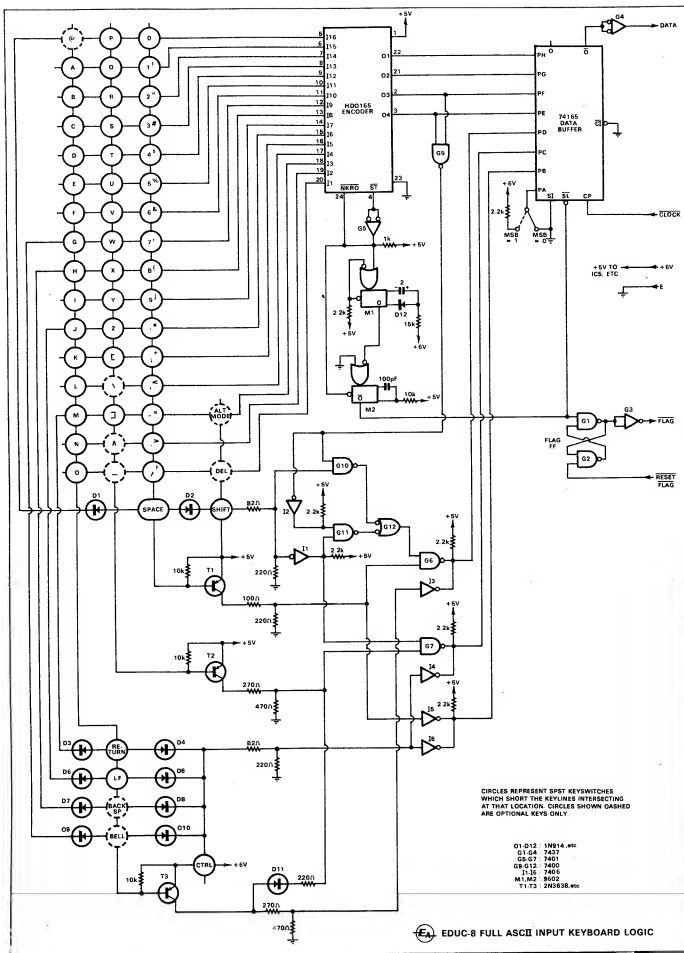
Similarly I5 and I6 are fed with the column 2/3 keyline signal from T1, and the control keyline signal, respectively, which ensures that bit 6 is false whenever a key or key combination corresponding to a character in columns 0, 1, 2 or 3.

Inverter I3 is fed with the column 4 keyline signal from transistor T3. As columns 0 and 7 are equivalent to column 4 characters combined with either the control or shift keys, this automatically ensures that bit 4 is always false for all column 0, 4 and 7 characters.

The signals fed to gate G6 are used to obtain the correct bit 4 encoding for columns 2 and 3. Because of the inverted shift mode operation of the row 12, 13, 14 and 15 keys, this is a little more complex than it might otherwise be. What must occur is that for the rows 0-11 keys, bit 4 must be false when the shift key is pressed, while for the rows 12-15 keys bit 4 must be false if the shift key is NOT pressed.

This is achieved by using gate G9 to monitor the bit 2 and bit 3 outputs from the HD-0165, so that its output is low whenever both these

*Shown opposite is the complete circuit of the ASCII keyboard encoder. Encoding of the 4-bit row address portion of the code is performed by the Harris HD-0165 16-line to 4-bit encoder IC, at top centre.*



## EDUC-8 computer

bits are high — signifying a row 12, 13, 14 or 15 character. This signal and its complement are then gated with the shift signal and its complement, by gates G10 and G11. The outputs of these gates are then combined by G12, performing an OR function. Finally G5 gates the resulting logic signal with the column 2/3 keyline signal from T1. If you trace all this through, you'll find that it does the required job. Monostable elements M1 and M2 are used to generate a clean "key pressed" signal from the strobe (L) and N-key-rollover (L) outputs of the HD-0165. In the configuration shown, they both suppress key contact bounce, and also prevent encoding errors due to accidental pressing of more than one key.

Gate G5 is an open-collector element, used as an inverter for the HD-0165's strobe (L) signal. When a single key is pressed, the output of G5 thus goes high, and carries with it the trigger input of monostable M1. M1 thus triggers, and its output goes high for a period of approximately 20ms. This interval is set by the 2uF capacitor and 15k resistor, with diode D12 used to compensate for possible capacitor leakage. The 20ms delay is to allow for keyswitch contact bounce to die away, before further events are initiated.

The Q output of M1 is connected to the complementary trigger input of monostable M2, so that when the output of M1 falls back to the low state at the end of the 20ms delay, M2 is normally triggered — producing a clean "key pressed" (L) signal at its Q-bar output. However because the reset input of M2 is connected to the output of G5, and both are tied to the "N-key-rollover" (L) output of the HD-0165, triggering of M2 can only take place if a single key remains pressed. If a second key has been pressed accidentally, the reset input of M2 will be held low, and it will be prevented from triggering.

If this occurs, correct triggering will take place when the second key is released, assuming that the intended key remains pressed.

The clean "key pressed" (L) signal produced at the Q-bar output of M2 is a pulse of approximately 500ns duration. The pulse is used to perform two functions, one being to enable the parallel-load input of the data buffer register — a 74165 device. This causes the encoded 7-bit ASCII character to be loaded into the buffer, ready for despatch to the computer.

The second function performed by the key pressed pulse is to set the flag FF, formed by cross-connected gates G1 and G2. Buffer gate G3 is thus able to take the flag line low, indicating to the computer that a character is available.

As with the other input peripherals described, the computer itself performs the actual data transfer, sending clock pulses to the data buffer register and accepting the data via the inverter/driver G4.

Note that as the EDUC-8 input/output circuitry is designed to handle 8-bit numbers, the keyboard data buffer is an 8-bit register. This means that there is a spare bit, as only seven are used for the ASCII encoded characters. The eighth bit may be set permanently to either a high (H) or low (L), as desired, by tying the PA input of the 74165 either to +5V (via a protective resistor) or to ground, as desired.

In many minicomputer systems the eighth bit produced by an alphanumeric input keyboard unit is permanently set to high (1), as this allows

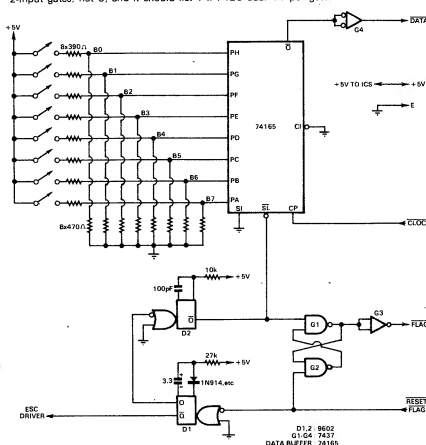
## MODIFIED TAPE READER LOGIC—CORRECTION

There is an old rule in electronics publishing, that it is very unwise to publish a circuit you haven't actually tried. It is almost inevitable that if you do so, the circuit won't work! I have proved the rule still applies, much to my embarrassment. In the description of punched paper tape peripherals, the circuit I gave in Fig. 6 showing how to modify the program-controlled reader logic for use with a motor-clutch type reader mechanism had not actually been tried. Due to lack of time, I had deduced it by analogy from Fig. 4.

Needless to say, it won't work properly. The main problem is that there is no way for the computer to know when the clutch mechanism has incremented the tape.

Rather than attempt to patch up this circuit, I have started again from scratch. And the effort was worthwhile, because it turns out that the job can be done much more simply, using only three ICs—a 74165 as the data buffer, a 9602 dual monostable for timing, and a 7437 for the flag FF and line drivers. The circuit is shown below, and it should be fairly self-explanatory. It should be taken to replace the original Fig. 6.

PLEASE NOTE ALSO that the parts list given for the program counter and address board (main computer parts list 3) is in error. It should list only 4 x 7400 or 9002 quad 2-input gates, not 5, and it should list 1 x 7420 dual 4-input gate.



SIMPLIFIED P.P. TAPE READER LOGIC FOR EDUC-8

ready identification of alphanumeric or "symbolic" character strings. I have used the same convention, and I suggest you do too.

In the prototype keyboard unit shown in the photographs, most of the logic was wired up on one of our multi-DIP utility PC boards. This was then mounted on the bottom plate of the keyboard case, using suitable spacers. Pieces of flat multi-coloured cable were used to connect the board to the keyswitches, and to transistors T1, T2 and T3. These were mounted for convenience on some existing tagstrips inside the main keyswitch frame.

Before using the keyboard assembly, it was necessary to remove all of the existing switch wiring as this was arranged in a connection array quite different from the one required. Before wiring the switches according to the new array, the complete assembly was cleaned

thoroughly and the key contacts sprayed with one of the aerosol contact cleaning fluids.

At the same time, the opportunity was taken to re-arrange some of the keytops, so that the keyboard became closer to that used in a typewriter or teleprinter. You may care to do this also, as the keys on many keypunch keyboards have the keys in rather different positions. The keys and positions of a normal ASCII keyboard are shown in the diagram, to help you if you want to aim for an arrangement as close to this as possible. I recommend that you do this unless you have a good reason to do otherwise.

As before, the keyboard unit connects to the computer via a length of multi-way cable, and a 6-pin DIN plug. As well as performing the logic signal connections, the cable also supplies the keyboard logic with its 5V DC power.

# Teaching your EDUC-8 to play a melody

As a change from the more conventional peripheral units so far described as part of the EDUC-8 system, the author here describes a "music player" output device. Producing musical notes which are under program control in terms of pitch, octave and duration, it may be used either purely for demonstration purposes, or to form the foundation of a digital music synthesiser system.

by JAMIESON ROWE

At this stage, having described a number of input and output peripheral devices of the conventional "computer attachment" variety, it seems appropriate to describe something rather less so. If nothing else, I hope this will perhaps start to show you that there is a whole area of largely unthought-of computer applications and interfacing possibilities, which is wide open for you to explore.

The device I have chosen to do this is a "music player" output device, which is capable of producing a monophonic melody under the direction of a suitable program in EDUC-8. While only a simple device, it provides program control of note pitch, octave and duration, and thus allows the computer to play tunes in quite a convincing manner.

As it stands, it is just the thing for demonstrating the EDUC-8 system to non technical people. Such people generally don't find the more conventional peripherals and functions of the computer very impressive, perhaps because it isn't easy for them to visualise just what is involved. But plug in the music player, feed in a simple tune playing program, and their eyes soon light up when the system starts playing "Greensleeves", or some other familiar tune—with no revolving record, unwinding tape, or other moving parts!

If you wish, however, it could be used as the start of a more elaborate arrangement. By adding facilities for programmable attenuation, filtering and so on, it could well be expanded into a digital music synthesiser system. The potential seems to be there for quite a lot of

development work along these lines, if you find the idea of digital music synthesis interesting.

Whether you intend building it up purely as a simple music player or as the start of a more complex synthesiser, I think you'll find the device well worth constructing. And good fun, too!

At the heart of the device is one of the new MOSLSI top-octave note frequency synthesiser ICs, which takes an input signal at the lowest common multiple frequency, and simultaneously performs all of the frequency divisions necessary to produce the 12 notes of the "even-temperament" musical octave.

There are two almost identical devices of this type currently available, either of which may be used in the player. One is the type AY-5-0212, made by General Instrument Microelectronics, and the other is the MK50242, made by Mostek Corporation. These are functionally equivalent, the only difference being that the latter operates from a single +12V supply rail whereas the former requires a -12V rail as well. The GIM chip is available on order from General Electronic Services, while the Mostek chip may be ordered from Total Electronics.

Whichever chip is used, in this circuit it is fed with an input signal at very close to 500kHz. As a result the 12 locked semitone outputs produced correspond to the octave beginning with the third C-sharp above middle C—i.e., the octave with A equal to 1760Hz. These notes are then fed to four cascaded binary dividers, so that the actual output pitch range of the

player unit covers the four octaves below this initially synthesised octave. In other words, the player produces output notes extending for two octaves either side of middle C.

In electronic organs, for which these top octave synthesiser chips are primarily intended, the input reference signal is generally derived from a crystal oscillator. This gives an appropriately high order of absolute pitch accuracy, as well as high stability. However for the present application a crystal oscillator has not been used, as high accuracy and stability are scarcely necessary.

Note, by the way, that we are talking here of absolute pitch accuracy and stability, not relative pitch accuracy. The intervals between the notes are automatically locked by the IC; it is only the absolute pitch of all the notes as a whole which can be varied.

As you can see from the circuit diagram, the oscillator used to generate the 500kHz signal is a simple R-C oscillator using half a 7413 Schmitt trigger device. This feeds the note synthesiser chip via a logic level converter stage using a BC108 or similar transistor. The oscillator frequency may be adjusted using the 25k pot, which thus becomes the "absolute pitch" tuning control for the player.

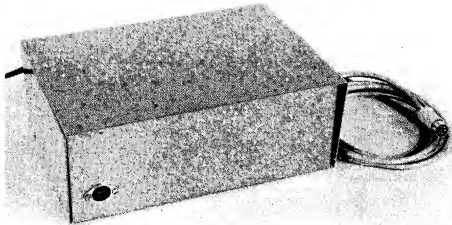
Strictly the exact oscillator frequency required will depend upon the absolute pitch reference used, because the IC division ratios used to give the various note intervals are only 3-digit approximations. Thus if you use A = 440Hz as the pitch reference, the correct oscillator frequency will be 499.840kHz, but if you use C = 261Hz, the correct oscillator frequency will be 499.032kHz. As you can see the differences are rather academic, and 500kHz is probably quite close enough for most purposes.

The 12 outputs of the note synthesiser IC are each provided with a buffer stage, using a BC108 or similar NPN transistor. This is again a logic level conversion, so that the signals may be fed to the TTL note multiplexer.

The note multiplexer is a 74150 device, which in effect forms a digitally programmable 16-position switch. Only one of its 16 inputs may be connected to the output at any one time, and the particular input selected is determined by the logic levels applied to the "address" inputs A, B, C and D.

As these inputs are connected in turn to the H, G, F and E outputs of the 74164 data buffer register, respectively, this means that the four least significant bits of the 8-bit number fed to the player from the computer provide the "instruction" to the note multiplexer. Each of the 16 possible combinations of these four bits

*Perhaps it doesn't look much like a music box, but hook it up to your EDUC-8 and an amplifier and it will play almost any tune you care to encode. This prototype was built up in a small utility case, with the ICs mainly on one of the EA "Multi-DIP" PC boards. The full logic diagram is shown overleaf.*





will force the 74150 device to connect the corresponding input to its output.

There are only 12 signals to be multiplexed, in this case, so that four of the 16 combinations provided by the 74150 are not used. These are 0000, 1101, 1110 and 1111, corresponding to octal 00, 15, 16 and 17, if specified by the computer data word, all four combinations produce silence, or a musical rest.

The 12 address combinations corresponding to octal numbers 01-14 inclusive are used for the active notes. The coding used is in simple progression, i.e., C-sharp is octal 01, D is 02, D-sharp 03, and so on up to C: 14.

The output of the 74150 note multiplexer is fed to a 7493 binary divider, which consists of four flip-flops connected in cascade. Thus the selected note frequency is divided by 2, 4, 8 and 16, with signals at the appropriate sub-multiple frequencies available at the four device outputs. The four 7493 outputs thus effectively provide the selected note, but in each of four octaves.

To select which octave is required, a 7401 quad 2-input gate is used to produce a simple 1-of-4 multiplexer. The four open-collector outputs of the 7401 are connected together, with a single 470-ohm output load. This forms a wired-OR configuration, so that any one of the four gates can feed the output.

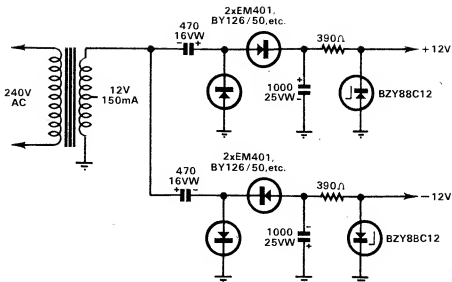
Gating signals for this simple octave multiplexer are derived from outputs C and D of the data buffer register, via half of a 9321 dual decoder device. Each half of the 9321 is a 2-bit decoder, with active low outputs. For each half of the device, only one of the four outputs can be low at any one time, and the output concerned depends upon the state of the two input bits. Thus the input bit combination 00 causes output 0 to go low, while input combinations 01, 10 and 11 cause outputs 1, 2 and 3 to go low respectively. Each half of the 9321 has a separate enable (L) input, which must be low before any output can go low.

As the 9321 device has active low outputs, and the 7401 device used as the octave multiplexer requires active high selection signals, a 7404 device is used for inversion. The two "left over" inverters in the 7404 are not used in the present player circuit, and could be used for other things if desired.

The output of the 7401 becomes the audio output of the music player, and is fed to an output socket via a simple R-C shaping circuit to provide a moderate amount of smoothing. This helps remove switching transients, and helps to make the square-wave signal a little less "harsh".

The audio output signal is of about 3V peak-to-peak, and has a source impedance of about 1k. You can therefore feed it into almost any audio amplifier, via the "radio" or "auxiliary" inputs.

At this stage, it should be fairly clear that the circuitry so far described is capable of taking the six least significant bits of a data number fed to the player from the computer, and using



A simple power supply circuit for the music player device, to provide the supply voltages not available from the computer itself. Note that the -12V section is only required for the AY-1-0212 chip, and may be omitted for the Mostek MK50242.

them to select and play any one of the 12 musical notes, in one of 4 octaves. Thus, for example, octal 22 will result in the sounding of the D in the second lowest octave, while octal 67 will produce the G of the topmost octave.

The remaining circuitry of the player device has been provided to automatically time the duration of notes. Four programmable note durations are provided, controlled by the remaining two bits of the data word—available at outputs A and B of the 74164 data buffer register.

The way in which the two control bits are used to control note duration is by having them determine the delay period of a simple timing circuit, used to set the flag flip-flop after the latter has been initially reset by the computer following delivery of the data word. By connecting the enable (L) input of the 9321 decoder feeding the octave multiplexer to the flag FF, this means that the note produced by the player only lasts for the interval between resetting of the flag FF by the computer, and the subsequent setting by the timer, after the programmed delay.

The timing circuit is basically the same as that used in the simple octal display unit, described earlier. It uses a D1371 programmable unijunction or PUT to discharge a capacitor, when triggered by the flag reset signal. The capacitor is then allowed to charge, and when it reaches a certain charge level a flag setting pulse is produced by a level detector circuit. The level detector uses two BC109 transistors, feeding the other half of the 7413 Schmitt trigger device.

The delay time is programmed by using four PNP transistors to switch in four different values of charging resistance. The transistors may be type 2N3638, BC178, BC327 or similar, and they are controlled by the four outputs of the second half of the 9321 decoder device, connected in turn to the A and B outputs of the note buffer.

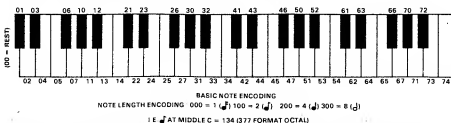
The four timing resistors have values which vary in binary ratio, so that each gives a delay time and note duration twice as long as its successor. Thus if the two most significant bits of the data word are both zero (octal 0-), the resulting note duration will be of say one unit long. If the less significant bit is 1 (octal 1-) the note will last for 2 units of time, while if the more significant bit is 1 (octal 2-) the note will last for 4 time units. Finally if both bits are 1 (octal 3-) the note will last for 8 time units.

The binary time ratios have been provided not because of some slavish desire to "keep everything binary", but merely because this is the normal relationship used in music. Thus octal 3- could correspond to a minim or half-note, octal 2- to a crotchet or quarter-note, octal 1- to a quaver or eighth-note, and octal 0- to a semiquaver, or sixteenth-note.

Naturally the correspondence between the binary/octal coding and the various notes will depend upon the actual delay time and the musical tempo. By varying the timing capacitor, shown on the circuit as 0.22μF, you can adjust the relationship at will. Larger capacitor values will give a slower tempo, or longer note values for the same tempo, while smaller capacitor values will give either a faster tempo or shorter note values.

You could provide a number of capacitor values, selected by a manual switch on the player, if you wish. Ideally the values should be in binary ratio, so that you could start with values of 0.1μF and 0.47μF in addition to the existing 0.22μF.

To recapitulate, then, the music player unit operates as follows. To play a note, the computer must send an appropriately coded data word into the player's data buffer register. The four least significant bits of the word cause the 74150 note multiplexer to select the desired note, while the next two bits are applied to the octave multiplexer half of the 9321 decoder, ready to select the octave in which the note



Using this note encoding guide, you should find it fairly easy to encode any desired tune for playing by the device. Note that the final code for a note is formed by adding the basic note code to the desired duration code, as shown by the example.

## EDUC-8 computer

is to be played

The computer must then reset the flag FF, which enables the 9321 decoder, and this in turn enables the octave multiplexer to begin sounding the note. At the same time the 01371 PUT in the timing circuit is triggered, discharging the timing capacitor. The second half of the 9321 decoder is also enabled, turning on one of the four timing resistor switching transistors according to the coding of the two most significant data word bits.

When the timing capacitor recharges, after the appropriate period of time, the level detector circuit sets the flag FF once more, disabling the 9321 decoder and hence causing the note to end. At the same time, the flag line of the player goes low, indicating to the computer that the note has been played. The computer may then send another note, if this is appropriate, to repeat the cycle.

Note that the player may be arranged to play "rests", or periods of silence, by feeding it words whose note coding corresponds to one of the four unused note multiplexer addresses. The duration of the rest may be programmed just as with normal notes, by using the two most significant bits. Thus octal code 100 will produce a rest of 2 time units duration, while code 300 will produce a rest 8 units long.

Note also that the 74164 data buffer register is fitted with an R-C circuit connected to the clear (L) input. This is to reset the register when power is first applied to the player, so that it doesn't burst into spontaneous song!

Most of the ICs in the music player are of the TTL type, and are powered from the 5V supply of the computer, via the connection cable. The only exception is the note synthesiser chip, where both of the alternative ICs require +12V, with the AY-10212 requiring -12V as well. To provide these voltages the player includes a small power supply, whose circuit is shown in the small diagram.

The current drain from the +12V rail is only about 20-25mA, while that for the -12V rail is even lower—around 4mA. As a result the power supply is very straightforward, using a miniature 12V/150mA transformer such as the Ferguson type PF2851 or similar. Two half-wave voltage doubler rectifiers are used, with simple zener diode shunt regulators. Note that the -12V part of the supply may be omitted if the Mostek 50242 note synthesiser chip is used.

I built up the prototype player unit in a small utility box, of the type used for small stereo amplifiers, etc. Most of the components and wiring were mounted on one of the EA "Multi-put" boards, which are very suitable for this sort of one-off project using a number of ICs.

The only parts of the circuit not mounted on the Multi-put board were the note synthesiser IC with its thirteen transistor buffer stages and power supplies and the power transformer. The latter was simply mounted in the case near the rear, with its primary connections taken to a 8-8 connector strip to mate with the mains cord wires in the approved manner.

To mount the note synthesiser chip, its buffers and power supplies conveniently. I used one of the PC boards originally designed for the "Crystal Locked Musical Tone Generator", described in the August 1974 issue of Electronics Australia. The board is coded EA 74/18, and was designed to take the chip and buffers, as well as power supply circuitry. Although the present voltage-doubler rectifiers

are a little different from those used in the original project, the power supply wiring can still be fitted on the PC board quite easily.

To give you an idea of the programming required for the music player, I have reproduced here a simple tune-playing program which will play a sequence of notes, automatically spaced by rests. Each rest is equal in duration to the note which immediately precedes it, a simple arrangement which gives a fairly natural sound to most tunes.

The coding for the tune to be played must be stored along with the program in the memory of the computer, beginning at location 030 octal. The tune can occupy the entire remainder of the memory, which forms the "tune buffer" area. As this comprises some 347 octal or 232 decimal locations, the program can play quite lengthy tunes. (Only the actual notes of the tune are stored, the program itself inserting the spacing rests.)

You can make the program play the notes of a tune without the spacing rests by replacing the instructions in locations 15, 16 and 17 (octal) with "NOP" instructions (octal 700). This will give a "legato" effect, which can be more appropriate with some tunes.

This is a very simple program, and you will no doubt want to try your hand at a more elaborate effort later on. But this one should at least serve to get your music player going.

To help you in encoding tunes for the player, we have prepared a diagram which shows the notes laid out in piano keyboard fashion, with their corresponding coding alongside. The note length encoding is also shown beneath; this is added to the note encoding to form the

complete code number for each note. Hence if you want a note to be middle C, and to last for 2 units of time, the coding is 100 + 34, or 134.

Note that the octal format assumed by the note encoding diagram is "377" format, which is fairly obviously the more appropriate in view of the way in which the player interprets the 8-bit words fed to it.

So that you needn't have to work out the encoding of a tune before you can get your player working, here is the coding for the familiar tune "Greensleeves". It is in 377 format octal, with bars separated by semicolons: 224; 327, 231, 233, 134, 233; 331, 226, 222, 124, 226; 327, 224, 224, 122, 224; 326, 222, 313, 224; 327, 231, 233, 134, 233; 331, 226, 222, 124, 226; 227, 126, 224, 223, 121, 223; 324, 224, 324, 200; 342, 242, 141, 233; 331, 226, 222, 124, 226; 327, 224, 224, 122, 224; 326, 222, 313, 200; 342, 242, 323, 233, 222, 124, 226; 227, 126, 224, 223, 121, 223; 324, 324, 200; 300, 300, 300, ... The code 300s at the end are simply to give silence after the tune ends, for better effect.

I suggest that you punch this coding onto paper tape, using a modified version of the program given earlier for program tape punching from the keyboard unit. Modify the program so that it will accept the octal input digits in 377 format, instead of the 737 format used for program encoding.

If you start this and all subsequent tune tapes with an "end of leader" bit code, followed by code 030, they can be loaded into the computer tune buffer at any time, without disturbing the tune playing program itself. This way, you can change tunes at will, without having to load in the program itself each time.

Incidentally, if you load in the tune playing program without an intentional tune, and set it going, it will "play" whatever random numbers are already in the computer's memory—either the remains of earlier programs, or the turn-on bias bits of the memory RAM flip-flops if you have just turned on. Either way, the sounds produced can be quite weird!

Well, that's the basic music player device, which will let your EDUC-8 play a simple tune. It's fairly straightforward, as you can see, and leaves plenty of room for elaboration should you be so inclined.

One idea would be to build a companion unit, which would take the raw audio output from the player and put it through a programmable attenuator and filter system, to vary both amplitude and harmonic content under the control of a second 8-bit word. Three of the bits could be used to give eight different amplitude levels, say, while the remaining five could be used to control the characteristics of a programmable formant filter.

For the attenuator you could use three NPN transistor switches, driven from the three appropriate outputs of the data buffer register, and switching the shunt resistors in a ladder-type resistive attenuator circuit. The filter scheme could be implemented in a similar way, with the transistors effectively switching high or low-pass filter sections in and out of the signal path. The whole unit might involve say a 74164 for the data buffer, eight 8C108 or similar transistors, and some R's and C's.

Another idea would be to expand the player itself, with additional data buffers, so that it could play a number of notes at once—for chords. The data words from the computer could be directed to the various buffers using a simple multiplexer scheme.

No doubt other ideas along these lines will occur to you as you go along. Have fun!

EDUC-8 PROGRAM			MUSIC PLAYER
			(LESS TUNE)
STEP	MNEMONIC		C.O.D.E.
0	PLAY, B		
1	1 MB		626
2	SWF		621
3	JMP, --1		582
4	JMP 1 PLAY		508
5	1N.W, B		808
6	BIFS, B		858
7	MASK, JMP	Y377 format	608
10	STT, CLA		718
11	TA7, BIFS		186
12	JCA 1N.W		386
13	GO, TA2 1 IN.W		105
14	JMS PLAY		488
15	TA2 1 IN.W		125
16	AND, MASK		887
17	JMS PLAY		488
20	IS2 1 IN.W		226
21	JMP 1 GOAL		525
22	NLT		721
23	JMP 1 STTA		504
24	STTA, B		818
25	GOAL, B		813
26	INDA, B		885
27			
30	(start of tune buffer)		
31	....		
32	....		
33	....		
34			
35			
36			
37			

This simple program will play tunes with each note followed by a rest of the same duration. The notes of the tune are stored in memory along with the program, beginning at location 030 (octal).

# Interfacing EDUC-8 to teleprinters & mag. tape

To conclude the description of his basic EDUC-8 microcomputer system, the author presents here a design for a flexible receiver-transmitter circuit capable of interfacing the computer with asynchronous peripheral devices such as teleprinters. Full details are given for interfacing with typical teleprinters, together with details of a simple system for magnetic tape recording using an elementary "modem"

by JAMIESON ROWE

No description of a computer system designed for educational work would be complete without at least a brief look at asynchronous interfacing. Apart from any other reasons, there is the very practical one that in most schools, colleges and universities the peripheral device most likely to be available for interfacing is the familiar teleprinter or "Teletype" (the latter name is a trademark).

I realise that for the individual private constructor, the interest in teleprinter interfacing is likely to be more theoretical than practical, as teleprinter machines are neither plentiful nor cheap. Brand new machines of the most appropriate 8-bit ASCII coded type cost anywhere from about \$850 to \$1500, depending upon options. Even secondhand machines in moderate condition tend to cost upwards of \$200, and they are not available very often.

Second-hand machines using the older 5-bit Baudot or Murray code are available slightly more frequently, and for a more reasonable cost. However while these can be interfaced with the computer, they are less attractive than the 8-bit ASCII type because of the need to arrange for code conversion.

Happily there is a second and quite practical application of the asynchronous interfacing technique, which should be well within the grasp of the individual constructor. Most people have a tape recorder, of either the reel-to-reel or cassette variety, and by using the asynchronous interface with a simple frequency-shift "modem" to be described, such recorders may be used for convenient storage of both programs and data.

But even if you don't have access to a teleprinter, nor feel disposed to build up the circuit for magnetic tape interfacing, I hope the discussion of asynchronous interfacing will prove interesting and worthwhile background information. It is after all an important area within the broad spectrum of computer interfacing, and as such is worth knowing about.

To begin, then. With all of the peripheral devices described so far, although the flag signals which initiate information transfer are generally not fixed in time relationship with respect to the computer's clock pulses, the actual transfer operation itself is always controlled by end locked to the clock pulses. In that sense they may be regarded as "synchronous" peripherals.

Inevitably, there arise situations where one wishes to interface a computer to devices which by their very nature do not lend themselves to this sort of synchronous transfer. Broadly speaking, the devices concerned are designed

to transfer information at their own fixed speed, which cannot readily be altered to synchronise with the computer. Probably the most common such device is the teleprinter machine.

Developed around 1906 as an improvement on the simple Morse key and sounder telegraph system, the teleprinter machine has a keyboard and printer mechanism resembling a typewriter. However unlike a typewriter the two are not connected permanently together; they are functionally separated, and share a common case purely for convenience.

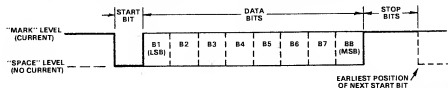
In the simplest possible telegraphy system using teleprinters, the keyboard of the machine at one end of the line is connected electrically to the printer of the machine at the other end, and vice-versa. However as this doesn't allow

type designed to deliver either 20 or 60 mA.

With no key pressed, the clutch is disengaged, and the distributor remains stationary. The brush contact touches a commutator segment permanently connected to the output line, so that the line receives current. This is known as the idle or "mark" condition.

When a key is pressed, a mechanical encoding system first operates a number of fixed switches. These open or close the connections between each of the various "data bit" segments of the commutator and the output line, setting up the coding of the character to be sent. Then the drive clutch is engaged, whereupon the motor rotates the distributor through one revolution. As it rotates, the brush contact effectively "scans" the various segments.

After leaving its idle position, the brush first contacts a segment which is permanently open circuit. This breaks the line circuit, to generate a no-current or "space" bit at the start of every character—i.e., the start bit. Then the brush scans the data bit segments, in each case making (mark) or breaking (space) the line circuit depending upon the coding set up for the character concerned. Finally the brush contacts one or more segments which are permanently connected to the line, to generate one or more



The serial data format used by 8-bit ASCII teleprinters, showing the way that the start and stop bits are appended.

each operator to see what they have transmitted, the connections are usually arranged so that each printer is also able to monitor or "echo" the information sent by its own keyboard.

Being designed for telegraphy, teleprinters are on-off or digital devices. And as they were developed for use over two-wire lines (or single wire and earth), they transfer the information in serial digital form. Each character is sent and received as a sequence of bits, with the number of bits per character and their transmission rate being fixed for a given type of machine and system.

Each character bit sequence has a fixed "start" bit, to identify the beginning of the character. This is followed by from 5 to 8 "data" bits, representing the actual character itself encoded in one of a number of codes. Finally there are one or more fixed "stop" bits, to identify the end of the character sequence.

The character bit sequences are generated at the teleprinter keyboard by a rotary commutator switch, known as the "transmitter distributor". This has a number of fixed contact segments, one for each of the total number of bits in the character sequence, and a rotating brush contact driven by a fixed-speed motor via a clutch. The rotating brush is connected to a power supply, generally a constant-current

mark bits—the stop bits.

At the end of the cycle the clutch disengages to bring the distributor to a halt, with the brush still in contact with the last stop bit segment. The machine thus stops with the line circuit made once more—i.e., in the mark condition.

The basic format of a teleprinter transmitted character thus consists of a start bit, a number of data bits, and a number of stop bits, where the start bit is always a "space" (no current), and the stop bit or bits are always "marks" (current). Many computer-type teleprinters use 8 data bits and 2 stop bits, giving the format shown in the small diagram.

The printer mechanism of the receiving teleprinter uses a similar technique to the keyboard in order to produce printed characters from the incoming serial bit sequences. There is again a rotating assembly driven via a clutch from a constant-speed motor, but in this case it is a set of selector cams which control the actuation of mechanical decoding linkages from an electromagnet. The electromagnet, known as the "selector magnet", is also used to control the driving clutch.

The signalling current flowing in the line is used to energise the selector magnet, and accordingly the magnet remains energised while ever the transmitting teletype is idling. In this situation the printer clutch is held disengaged.





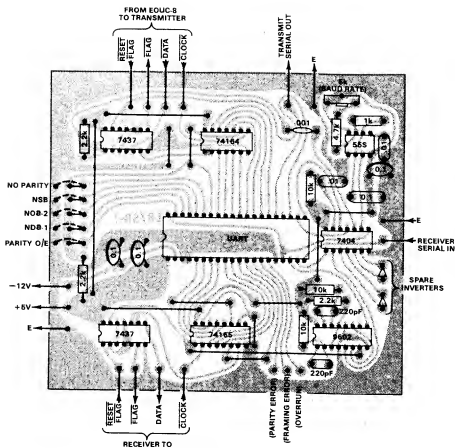


FIG. 2 ASYNCHRONOUS INTERFACE PCB WIRING

mechanism from a computer. As the printing mechanism is already designed to automatically synchronise with incoming characters, it will operate normally.

However it is when one comes to consider using the teletypewriter keyboard to send characters to the computer that the second problem arises. This is because the keyboard cannot be synchronised with the computer; it sends each character bit sequence at its own rate, and immediately following a keystroke.

What is needed for keyboard interfacing is therefore a logic circuit capable of electrically duplicating the operation of the teletypewriter selector mechanism—one which is able to recognise the arrival of a start bit, and synchronise with it to strobe in the following data bits.

This is by no means an easy task, as a few moments' reflection may reveal. Until a few years ago, teletypewriter keyboard interfacing thus involved quite a lot of complex circuitry.

Happily there are now available single LSI integrated circuits which take care of the whole operation of interfacing with asynchronous devices like teletypewriters—both the transmitting and receiving sides. They are called "universal asynchronous receiver-transmitters", or "UARTs" for short. The word "universal" is used because the devices are arranged to be programmable in terms of the number of data bits, stop bits, baud rate, and so on, to suit a wide variety of teletypewriters and other devices.

UARTs generally come in an impressive 40-pin dual-in-line package, and considering their complexity they come surprisingly cheap—around \$8-10. This seems to be because they are used in large quantities in the data communications industry.

As you can see from Fig. 1, a UART is used as the heart of the asynchronous interfacing

unit I have developed for the EDUC-8 system. And using such a device, the interface becomes deceptively simple and straightforward. There are three alternative UART devices which may be used: the American Microsystems S18B3 (available from Cema Distributors), the General Instrument Microelectronics Ay-5-1012 (from General Electronic Services), and the Signetics 2536 (from Tecnico Electronics).

Strictly the three devices are not quite identical, because the S18B3 is capable of being programmed for "1.5" stop bits, in addition to the single and double stop bit programming provided on the other two devices. However this is really only an academic difference, as it only affects the maximum rate of character transmission when working with 5-bit teletypewriters. All three devices will work with such machines, as well as with most others, and thus for all practical purposes they may be regarded as identical.

We don't have the space here to enliven the operation of a UART in detail, and if you are interested I suggest you try and get hold of a data sheet and applications brochure from one of the distributors. However if you bear in mind the foregoing description of teletypewriter operation, the general idea should become clear to you as we look at the circuit of Fig. 1.

The transmitter and receiver sections of the UART both need a source of external clock signals, at a frequency of 16 times the desired baud rate. Thus for normal 110 baud teletypewriters, the required clock rate is 1760Hz. As you can see, this is provided by a simple pulse generator using a 555 timer IC. The 5k preset pot is used to set the frequency to produce the exact baud rate required. The values of the pot, its series resistor and the 0.1  $\mu$ F charging capacitor may all be changed, if necessary, to adapt

the interface to baud rates very much higher or lower than the nominal 110 baud rate shown.

A single set of logic inputs are used to program both the transmitter and receiver sections of the UART for the serial word format required. The inputs are those marked NPB (no parity bit), NSB (number of stop bits), POE (parity odd/even), NDB1 and NDB2 (both used for setting the number of data bits). These are taken to either the high or low logic level to program the device for the format required.

For our purposes, the most appropriate format is words having 8 data bits, no parity, and 2 stop bits. This is very suitable for handling both instruction and data words from EDUC-8, and is also the format used on many 110 baud teletypewriters. As it happens, this format is programmed by taking all five of the UART logic inputs to the logic high level, as shown.

Probably the only other format you are likely to want is that for 5-bit teletypewriters, which use 5 data bits, no parity, and "1.5" stop bits. To program the UART for this format, simply take the NDB1 and NDB2 inputs down to low level, leaving the others at high level. Strictly only the S18B3 device will give the correct "1.5" stop bits, but the other devices will still give satisfactory operation.

The UART transmitter section has 8 parallel inputs for the data bits to be transmitted, labelled TD1-8. To provide these with the data word to be transmitted, the interface uses a 74164 device as a buffer. The word is shifted into the buffer serially, at the computer clock rate, as with the other output devices which have been described.

The word is actually loaded into the UART's internal data buffer from the 74164 by the RESET FLAG (L) pulse from the computer, which is fed not only to the transmitter flag FF, but to the TDS-bar (transmit data strobe) input of the UART. This input not only causes the word to be fed into the UART, but also triggers the transmitter circuitry to begin transmitting it. Accordingly the UART adds the start bit and eight data bits, and feeds the word out of the TSO (transmit serial out) terminal, at the correct baud rate.

When the last data bit has been transmitted, the UART signals that a new number may be loaded for transmission, by producing a high logic level at its TBMT (transmitter buffer empty) output. This signal is fed through inverter I2, and used to set the transmitter flag FF via the .001  $\mu$ F differentiating capacitor. The flag FF is formed by gates G2 and G3, with G1 used as a buffer for the FLAG (L) line to the computer.

When the UART sets the flag FF to signal that the last data bit of the current word has been transmitted, the stop bits still have to be transmitted. Thus the computer has a time period equivalent to two bits at the asynchronous rate—i.e., about 18 milliseconds at 110 baud—in which to feed the next number into the 74164 buffer. The computer can thus keep up with the UART quite easily, and ensure characters to be sent at the maximum character rate, if desired.

The receiver side of the UART accepts the asynchronous serial input at its RSI (receiver serial input) terminal. When a character arrives at this terminal, the receiver circuitry automatically detects its start bit, synchronises with it, and strobes the following data bits into an internal buffer. It then indicates that a received word is available at its eight parallel outputs RD1-8, by producing a logic high level at its ODA (output data available) output.

The external receiver interfacing circuitry inverts this signal through inverter I3, and uses it to trigger a monostable, D1. The output from D1 is a narrow pulse (about 1  $\mu$ s), and is used

### EDUC-8 computer

to load the data from the UART outputs into the external receiver buffer register, formed by a 74165 device. The pulse is also used to set the receiver flag FF, formed by gates G5 and G6. Gate G7 is used as before to provide buffering for the FLAG (L) line.

The output of monostable D1 is also fed to a second monostable D2, whose output is a second 1  $\mu$ s pulse immediately following the first. This is used to reset the UART's internal receiver flag circuitry, readying it for the arrival of another character.

When the receiver flag FF is set, the computer is thus made aware that a character has been received and is ready for transfer in the receiver buffer. It can then transfer the character into the AC register as with any other input device, by supplying clock pulses and receiving the data via the line driver G8.

Note that the UART requires both 5V and -12V power supplies, being a MOS device. The 5V supply comes from the computer, and is used to power the other interface ICs as well. The -12V supply involves only about 40mA, and is provided by a small supply in the interface unit itself. Inverter I4 and its associated R-C network are used to reset various internal UART circuitry when the power is first applied.

To make it easier for you to build up the basic asynchronous interfacing circuit of Fig. 1, I have designed a small PC board for the job. It is coded E8/SR-T, and the wiring for the board is shown in Fig. 2. As you can see, links are provided to allow the unit to be programmed for various word formats.

Incidentally, you probably realised it anyway, but just in case you haven't I should perhaps point out that the interface connects to two of the EDUC-8 input/output ports. The transmitter section connects to one of the output ports (OD0 or OD1), while the receiver connects to one of the input ports (ID0 or ID1). This is because it is really two peripheral devices in one—or more correctly, it "interprets" for two.

The TSO and RSI terminals of the UART are not designed to connect directly to a teleprinter line. In fact they are basically standard TTL logic terminals, with the convention such that the logic high level represents the "mark" condition for both, and logic low the "space" level. Both normally remain high when the UART is idling.

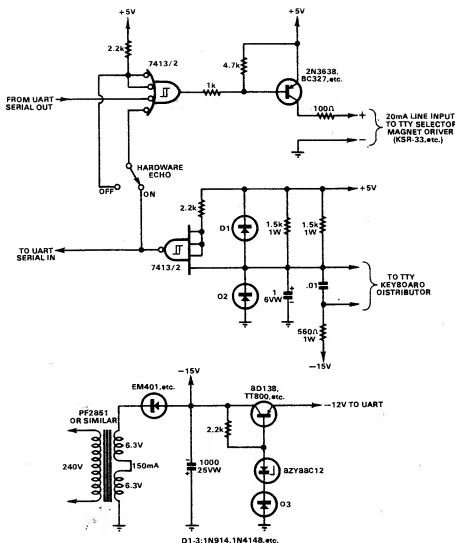


FIG. 3 TELEPRINTER INTERFACING (33 SERIES TELETYPE)

To connect the interface with a typical modern teleprinter like the Teletype series 33 machines (KSR-33 or ASR-33), you will need supplementary circuitry as shown in Fig. 3. This also includes the small power supply required to produce the  $-12V$  rail for the UART.

The 33 series Teletypes have an internal selector magnet driver circuit, so that the incoming signal line does not drive the magnet coils directly. This simplifies the external circuitry required, as there is no need to worry about inductive spikes, back-EMF, etc. All that is required is a logic inverter and a simple PNP transistor stage, shown at the top of Fig. 3.

Note that as shown, the PNP stage is designed to provide 20mA of signal current into the teleprinter selector magnet driver. This of course assumes that the driver is adjusted to expect this current level, rather than the alternative 60mA level. The changeover from 60mA to 20mA input is quite easily made, if required, by swapping over a link in the driver circuit.

The remaining logic shown in Fig. 3 forms an input shaping and Schmitt trigger circuit to produce a signal from the teleprinter keyboard distributor contacts, suitable for feeding to the UART serial input on the interface PC board. The R-C components and diodes are used to clean up the signal fed to the 7413 Schmitt trigger element, to remove bounce and other spurious transients.

Note that the distributor contact circuit is connected to both the 5V supply rail and to a source of -15V from the small power supply. The resulting 20V swing helps in producing a clean signal from the rotating distributor contacts.

You have probably noticed that the second half of the 7413 Schmitt trigger device is used

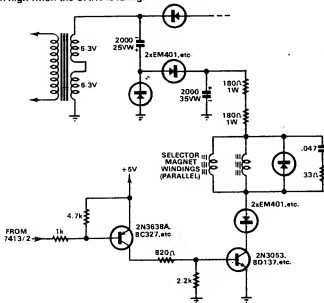
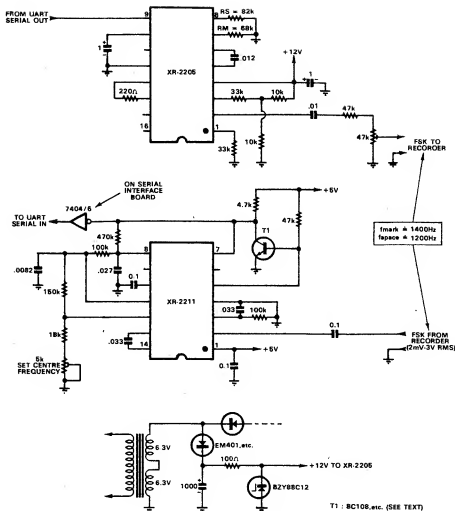


FIG. 4 SELECTOR MAGNET DRIVER FOR OLDER TTYS



es the inverter for the PNP selector driver amplifier. This makes it possible to provide a "hardware echo" facility, so that if desired the printer can be made to automatically echo whatever is transmitted to the computer from the keyboard. As you can see, this is achieved simply by disconnecting one of the spare inputs of the 7413 from logic high, and connecting it instead to the output of the lower 7413 element. The top element thus becomes a negative-input OR gate, feeding signals to the printer from either the UART or the keyboard.

Older teleprinter machines differ from the newest 33-series Teletype machines in that they generally do not have an internal selector magnet driver circuit. In other words, the incoming signal line drives the selector magnet windings directly. The magnet has two windings, which are connected in series for 20mA operation or in parallel for 60mA operation.

To operate one of these older machines with the interface, you will need to modify the circuit of Fig. 3 as shown in Fig. 4. The change mainly involves the addition of a further rectifier circuit to the power supply, to generate about 25V, together with a medium power NPN transistor to switch the selector magnet current.

As you can see, the magnet windings are connected in parallel for 60mA operation, as this gives more reliable operation from a 25V supply. The two 180 ohm resistors are to set the current level, while the two diodes and the R-Circuit associated with the magnet windings are to suppress the inductive back-EMF.

Don't forget that in order to operate an older teleprinter of the 5-bit variety with the EDUC-B

system (or with any other computer, for that matter), you will have to perform code conversion somewhere in the system. This is because 5-bit machines use the Baudot or Murray code, not ASCII.

The code conversion could be done by the computer program itself, but this will of course involve valuable memory space. Perhaps a neater way would be to interpose read-only memory circuitry between the UART data inputs and outputs and the 74164 and 74165 buffers, in the interface circuit of Fig.1. You could use either IC ROMs, or diode arrays.

I hope the foregoing information will enable you to connect up the teleprinter to your EDUC-8 system, if you want to do so, with a fair degree of confidence and success.

As mentioned earlier, the esynchronous interfacing unit of Figs. 1 and 2 may also be used for storing program and data on magnetic tape. Before closing, I will give a brief description of how this is done. Almost any mono recorder may be used, of either the cassette or reel-to-reel variety.

As with a teleprinter, it is not all that difficult to transfer information from the computer to magnetic tape—providing the transfer is made at a suitable rate, in this case one which will fall within the modest bandwidth of an audio recorder. The problems tend to occur in the reverse transfer direction: from the tape back to the computer. Like the teleprinter, a tape recorder tends to supply information at its own fixed rate, and cannot be synchronised readily with the computer clock pulses.

Happily the same asynchronous date format

used for teleprinters may be used for tape recording and playback—all that is needed is to encode the data in audio tones, so that the recorder is not required to handle DC levels.

The most reliable results are obtained using the technique of frequency-shift keying, or "FSK", where the two digital data levels are recorded as tones of differing pitch. This means that the tape is always recorded with a tone of one pitch or the other, which allows the effects of drop-out to be minimised.

Generating and demodulating FSK signals used to be a fairly complex business, but thanks to modern IC technology it is now fairly straightforward. As shown in Fig. 5, only two ICs and a transistor are required in order to adapt the basic asynchronous interface circuit of Fig. 1 for magnetic tape recording.

Both of the ICs are made by the Exar Corporation, and are available in Australia from A. J. Ferguson Pty Ltd (order through your usual supplier). They are the XR-2206 waveform generator device, and the XR-2211 tone and FSK detector device. Each costs around \$5.00.

The XR-2206 device is designed to produce either square, triangular or sine waveforms over a wide frequency range, and has the additional feature that its frequency may be switched between two values by a TTL logic signal applied to pin 9. Here it is used to generate sine waves of either 1400Hz or 1200Hz, with the higher frequency corresponding to 'mark' and the lower to 'space'. The signal from the TSO output of the UART is fed to pin 9 to produce the required FSK signal, which appears at pin 2 of the device. This is fed to the recorder via a preset pot, to adjust recording level.

To decode the FSK signals on playback, the output from the recorder is fed to the XR-221 device. This device is especially designed for FSK demodulation, and works on the phase-locked loop principle. It will lock onto input signals anywhere between 2mV and 3V RMS, so that the output from the recorder can vary over a very wide range without causing a data error.

The XR-2211 device has two outputs, a tone-detect (L) output which appears at pin 5, and the actual FSK demodulation output which appears at pin 7. By wiring transistor T1 as shown, the output from the device remains low when there is no incoming tone—i.e., when the tape is stopped. By using one of the spare inverters on the interface board, this low is changed to a high, so that the RS1 input of the UART receives the correct "mark" signal whenever a character is not actually being played back from the tape.

As soon as the tape is started and tone appears, the level at pin 5 of the XR-2211 falls to the low logic level, and transistor T1 turns off. However the level at pin 7 will still be low while the tone from the tape is at the "mark" frequency. It will only go high when the tone changes to the "space" frequency, and the inverter will thus feed the correct logic levels to the UART.

For correct operation the VCO of the XR-2211 must be set to give a free-running frequency midway between the incoming mark and space frequencies—i.e., 1300Hz in this case. This is set by means of the 5k preset pot in series with the 18k resistor, connected to pin 12.

You can perform this adjustment reasonably well by recording a section of tape with a single character repeated, and monitoring the output signal going to the RSI input of the UART with an oscilloscope. Then adjust the 5k pot while playing back the tape, until you are getting a clean, noise-free signal.

(Continued on page 107)

## INFORMATION CENTRE

(Continued from p.105)

It would be more convenient if editorial pages were separated by a full page advert, in practice this situation is virtually impossible to achieve. We would refer you to the Forum pages of the January 1974 issue of EA, where we answered a previous critic in some detail. Thank you for your suggestion and your kind remarks.

**LOW FREQUENCY RESPONSE:** I am 15 years old and have been interested in electronics for about 2 years now. I must congratulate you on a clearly set out and informative magazine. I have received the PM 136 amplifier and thanks to the simple instructions and clear diagrams managed to get it operating first go.

I connected the amp to a turntable fitted with a suitable cartridge and a pair of speakers, and I am extremely pleased with the resulting sound.

As I operate the set in a very small room (about 12' x 9'), it requires a fair degree of bass lift to achieve a "flat" sounding response. This, however, dramatically increases turntable rumble as well as low frequency surface noise to a very large extent, causing the woofer cone to travel about 2" excursions at half volume setting.

Is there any way of attenuating the response below about 20Hz without major circuit changes to the amplifier? It seems to me that this must be quite a common problem, so I am hoping you can answer this in the pages of "Information Centre". (N.G., Carlingford, NSW.)

Thank you very much for your comments about the magazine. It is possible to design circuits to roll off the bass response at 20Hz, although we have never presented such a design. From the description of your symptoms, it appears that you may be suffering from acoustic feedback between the speakers and the turntable unit.

A simple test for this is to move the turntable to another room. If the large cone excursions stop, it is almost certain that acoustic feedback was the cause of the excursions. It will now be necessary to return the turntable to the original room, while still retaining the acoustic isolation.

It may be necessary to increase the isolation by providing extra suspension for the plinth, or by placing it on some sort of flexible mounting, similar to that used to support the turntable on the plinth. Another approach is to use a wall shelf to support the plinth and turntable, to avoid coupling of vibrations from the floor.

If acoustic feedback is not causing your troubles, then there is little which can be done. A high pass filter with a cutoff frequency at 20Hz will not eliminate any audible rumble and surface noise. However, it may eliminate distortion caused by bottoming of the woofer cone due to large subaudible signals.

Any filter with a higher cutoff frequency will eliminate rumble and surface noise, along with any signals which have similar frequency components, and for this reason is not recommended. We can only suggest that it may be necessary to use less bass boost.

## EDUC-8 computer—from page 73

For slightly better results, it is necessary to measure the actual output frequencies of the XR-2206, using a digital counter. These will probably be slightly different from the nominal figures, due to the effect of component tolerances. Then work out the mean of the two frequencies, by adding them together and dividing by two.

Finally, set the free-running frequency of the XR-2211 to this mean frequency, in the following manner. With the power off, disconnect the 0.33uF capacitor from pin 3, and connect a temporary link between pins 2 and 10. Then turn on the power, and with no tone input signal, connect the digital counter to pin 3.

**LOST IN CAVES:** An idea occurred to me while listening to the current record "Journey To The Centre Of The Earth" concerning a possible project. Two people had become separated while exploring a cave but were able to find each other and determine their distance apart "by use of their chronometers". I expect this was done by measuring the time lag between their speech.

I have thought out a way in which this idea could be implemented (details enclosed), and was wondering if this could possibly form the basis of a project. (Tony Picone, 99 Rowena Street, Richmond, Vic. 3121.)

Due to the rather specialised nature of the device you have in mind, we do not think that it would be suitable as a project. However, we will keep your idea in mind. We have published your full name and address so that any interested readers can contact you direct.

**DIMMER BUZZ:** Recently, I had an opportunity to tape an amateur stage show at a nearby hall. Lighting for the stage was dimmed by modern solid state triac control using all three phases of the electricity supply. The result was a quite troublesome buzz throughout the whole length of the recorded programme except for the start when the lights were not working. The microphone leads were 12 yards long, of shielded cable.

My questions are: Was the interference induced through the input to the tape recorder via the microphones and long leads or via the mains supply which the tape recorder and light dimmers were connected to, or both of the above? How can I eliminate the problem on future occasions? Thanks for an interesting magazine which is well worth the price. I never miss a copy. (T. P., Richmond, Vic.)

It is quite likely the buzz you experienced was radiated via the supply lines as RF interference which could be induced directly into the mic cables or picked up directly by the tape recorder. In addition, the mains interference could be fed directly into the tape recorder via its power supply. Without being in a position to eliminate each possibility, it is not possible to nominate the principal cause. As a first suggestion, 12 yards of cable is rather long unless you are using low impedance microphones and/or balanced input connections.

**DEAD LETTER:** We are holding material originally addressed to Mr D. Shropshall, PO Box 216, Roebourne, W.A. 6718. This has been returned by the postal authorities, presumably because they could not deliver it. If Mr Shropshall will advise us of his present address we will forward the material to him.

(Continued on p. 110)

## NOTES & ERRATA

**PHILIPS 10GHz DOPPLER MODULE** (May 1975): The circuit of the prototype intruder alarm should show the 100k load resistor of the BC109 preamp transistor connecting to the 8.2V supply rail, not to the 10V rail.

Adjust the 5k pot until the counter indicates the correct mean frequency. Then turn off the power, and restore the circuit as it was. The XR-2211 should now give very clean and reliable demodulation of the FSK recordings.

Note that the XR-2206 device requires a 12V power supply rail, so that a small rectifier circuit must be added to the interface power supply as shown. The XR-2211 operates from the 5V computer rail.

In closing, it might be worthwhile to point out that the simple FSK modulator-demodulator or "modem" of Fig. 5 could also be used for transmitting data signals over radio links. ☺

# jaycar

PTY. LTD.

## ANNOUNCE THE ARRIVAL OF A BULK SHIPMENT OF DUKE AM/FM/MPX STEREO TUNERS

### Specifications:

FM—88-108MHz

S/N ratio better than 50dB  
IF rejection better than 90dB  
Stereo separation better than 30dB  
Muting and AFC switches  
Linear scale dial  
Signal strength meter  
FM stereo auto mpx.

AM—535-1605kHz

Sensitivity 50dB  
IF rejection 45dB  
Selectivity 25dB  
Ferrite bar antenna  
Flywheel tuning



## DUKE A100 STOCK AVAILABLE THIS MONTH

### WHOLESALE ENQUIRIES TO:

John Carr & Co Pty Ltd  
405 Sussex St,  
Sydney.  
Phone 211-5077

### RETAIL ENQUIRIES TO:

Jaycar Pty Ltd  
PO Box K39,  
Haymarket 2000.  
Phone 211 5077.

## INTERFACING A BURROUGHS SELF-SCAN DISPLAY PANEL TO YOUR EDUC-8

A Burroughs Self-Scan display panel is very suitable for use with the EDUC-8 computer system, offering full alphanumeric readout with large, brightly lit characters. The interfacing needed to hook up such a panel to your machine is quite straightforward, as explained below.

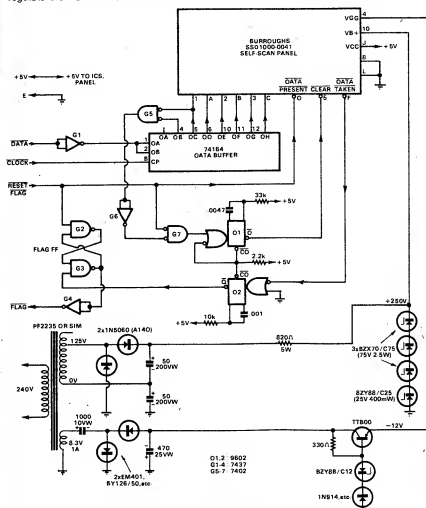
There are a number of display panels in the Burroughs range which could be used with the EDUC-8 system. I elected to try the SSD1000-0041, which has an internal refresh memory and can display a line of 16 characters 10mm high. Alternatively you could use the SSD1000-0040, which can display a line of 32 characters about 5mm high. There is also a panel which will display up to 80 characters 5mm high. All panels are available from Cema Distributors Pty Ltd, of 21 Chandos St, Crows Nest NSW 2065.

The SSD1000-0041 costs \$204, plus \$2 for a matching 10-way PC connector (SR-127), and 15% sales tax where applicable. This seems quite reasonable considering that it offers inbuilt refresh memory. Cema are providing data sheets with each unit purchased.

For details on the operation of Self-Scan discharge panels, I can only refer you here to Greg Swain's article in the October 1974 issue of Electronics Australia.

The interfacing required for the 16-character panel is shown below, and as you can see it is quite straightforward. A 74164 device is used as a data buffer, to provide the panel's required 6-bit ASCII characters. The RESET FLAG(L) pulse is used to enable the panel's DATA PRESENT (L) input, while the panel's subsequent DATA TAKEN (L) output is used to set the flag FF via monostable D2—used to stretch the pulse to ensure overlapping the input pulse. Gates G5, G6 and G7 and monostable D1 are used to clear the display whenever a carriage return or similar non-printing control character is received, to begin a new line.

The Self-Scan panel requires two additional supply voltage apart from the +5V from the computer: +250V for the actual display, and -12V for the MOS refresh memory. These are provided by the simple power supply shown. Note that zeners are used to regulate the 250V line, which must be maintained within plus/minus 5%.



INTERFACING FOR BURROUGHS SELF-SCAN PANEL

## printed circuit

- Accurately machine printed / etched.
- Phenolic & fibreless-gold-plated.
- Special manufacturers' packs of 10.
- EA, R&H, ET, Philips, Mullard available.
- Specialties to your drawing.
- Plus post 40c. 1mm despatch.

ET1532A.8	2.50	ET801N	2.60	72720	2.20
ET1540	4.50	ET801E	2.80	ET823	2.80
ET1545	2.50	ET802	2.20	ET827	2.80
ET1547	2.00	740PL	2.00	7272ABC	4.40
ET1552	2.20	73172.80		73010	3.30
ET1553	3.30	ET801N	2.20	73014	6.50
ET1555A	4.50	ET801C	2.80	725A1	3.30
ET1562	2.50	ET420C	2.40	ET822	2.80
ET1618	2.80	ET4200	2.40	71C12	4.40
ET801P	2.50	ET4208	2.80	ET819	2.80
ET801T	2.00	ET420A	2.20	ET818	2.80
72E6M	2.50	ET524	2.80	ET817	2.80
75054	2.50	ET801E	3.30	ET814	2.50
75A01	2.50	ET801M	3.80	ET807	2.50
ET1414E	2.80	ET801F	2.80	71112	3.30
ET141402	3.20	ET801E	3.80	ET811	3.80
ET143	2.80	ET801	3.30	719P	2.80
ET14	2.00	737011	2.80	ET808	2.80
ET116	2.80	739P	2.20	ET823	2.80
ES5	3.00	ET520A	4.40	ET812	2.20
EN1	3.00	ET520B	4.40	7265	2.50
EN2	4.00	72512	4.80	ET805	2.50
ET1312	3.00	738A9	2.80	ET806	2.80
7601	2.50	ET113	3.10	ET824	2.50
74M1220	3.50	ET418	2.20	715A4C	3.30
74M122C	2.80	ET218	3.50	715A4B	2.80
74M12	3.20	ET417	1.70	715A4A	2.80
74M12A	2.80	ET309	2.80	ET825	2.80
ET801	2.50	ET4140	1.20	71W16	2.20
ET802	2.80	73107	2.80	71W1A	2.20
ET428	2.80	7358	1.30	ET803	2.80
ET131	2.80	ET821	3.90	7163	3.00
ET132	2.80	ET212	1.10	ET804A	2.50
ET427	2.80	ET418	3.30	71122	2.20
ET428	2.80	7381	1.70	725L1	4.00
74M08	2.00	ET518	2.20	70PA1	4.00
74E8M	2.50	733C	2.80	70M01	4.00
ET418	2.50	7311	2.80	70T2	4.00
ET428	2.20	ET414C	2.80	70F10	4.00
EX1	5.00	ET414B	2.80	70A2	4.00
EX10T	5.00	ET414B	2.80	ET804	2.50
EXP	5.00	72M12	2.80	70C01	2.20
74E8M	5.00	725A9M	2.80	7181	2.20
EBD	5.00	ET413	2.80	70C7	2.00
ERA	5.00	ET834A	3.10	70P5	4.00
EST	6.00	72511	2.80	71A8	2.50
ETC	6.00	7257	2.80	70M5	4.00
74C93	4.20	72110	1.30	70B5B	2.20
7408	4.00	ET827-40	0.00	70X11	8.00
ET414	2.60	725A10	3.10	70C4	8.00
ET1311	2.50	725C	2.80	70C1	4.00
ET828	2.50	ET828	2.20	70P1	2.50
ET114	2.50	72510	2.80	88C11	8.00
ET808	2.80	7288	2.80	88F10	2.80
74045	4.00	725A9	2.80	70A1	2.50
ET801M	2.50	ET803	3.30	88F10	8.00
ET801L	2.80	72M3M	3.10	88C10	3.00
ET422	3.30	7273	3.30	88P0	3.00
74053	2.50	ET828	2.80	88C0	3.00
ET801J	3.00	72M4	2.20	88P5	3.00
ET423	2.20	71A4	2.80	8810CL	8.00
ET420E	3.30	72P3	2.20	88S3	6.00
ET8218	2.00	7282	2.80	88T5	4.00

## ALL SILICON 30/60w PA PORTABLE AMPLIFIER



## COILS and IF's

10" x 6" x 2 1/2" H. All £2.50 ea. plus post 30c.

RF CHOKES Plus post 40c.  
381 AIR: 2.5mH 50mV Pwr 70c.  
381 IRON: 10uH to 1,000uH 25ma 70c.

## FILTERS

Plus post #1.  
27: Line filter 2 amp #12.  
29: Line filter 10/20 amp #37.80.  
30: Pulse filter 2 amp #12.

MAIL cheque or money order  
(add postage) direct to:—

**RCS radio pty ltd**  
651 FOREST RD BEXLEY  
NSW 2207 587 3491