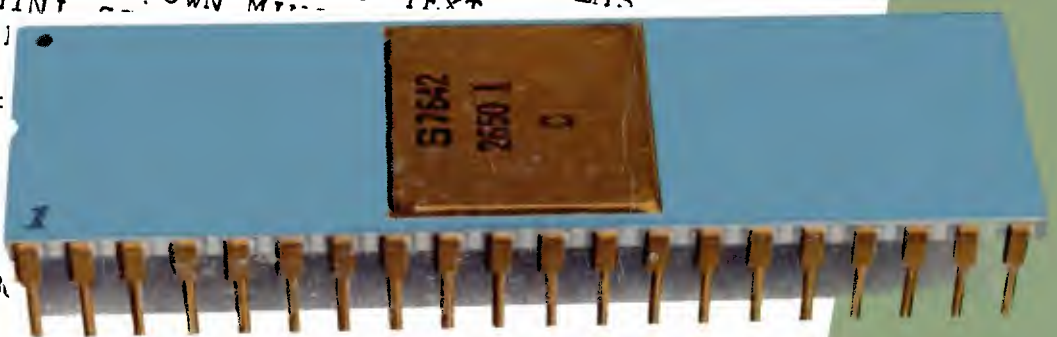
GETTING INTO MICROPROCESSORS

$4·50

1 GETTING INTO MICROPROCESSORS
2 MICROPROCESSORS: THE BASIC CONCEPTS
3 INTEL'S 8080A CHIP & SDK-80 EVALUATION KIT
4 MOTOROLA'S 6800 CHIP & MEK6800D2 KIT
5 NATIONAL SEMICONDUCTOR'S SC/MP CHIP & SYSTEMS
6 NATIONAL SEMICONDUCTOR'S PACE CHIP & SYSTEMS
7 SIGNETICS' 2650 CHIP & EVALUATION SYSTEMS
8 FAIRCHILD'S F8 CHIP SET & DESIGN EVALUATION KIT
9 TEXAS INSTRUMENTS' LCM-1001 MICROPROGRAMMER
10 MOSTEK'S F8 CHIP SET & EVALUATION KIT
11 THE LEAR SIEGLER ADM-3 VIDEO TERMINAL KIT
12 THE MINI SCAMP MICROCOMPUTER 1: BASIC DESIGN
13 THE MINI SCAMP MICROCOMPUTER 2: PCB & CASE
14 MAKING MINI SCAMP BIGGER AND BETTER
15 MORE ON MINI SCAMP: IMPROVEMENTS
16 A "BABY" SYSTEM USING THE SIGNETICS 2650
17 VIDEO DATA TERMINAL FOR MICROCOMPUTERS --1
18 VIDEO DATA TERMINAL FOR MICROCOMPUTERS --2
19 CASSETTE TAPE INTERFACE FOR MICROCOMPUTERS
20 THE NRZ RECORDING TECHNIQUE
21 SIMPLE POWER SUPPLY FOR MICROPROCESSOR SYSTEMS
22 MICROPROCESSOR-BASED ASCII-BAUDOT TRANSLATOR
23 ADDING A CURSOR TO OUR VIDEO TERMINAL
24 SC/MP GETS TINY BASIC
25 A TEXT EDITOR FOR SC/MP LCDS SYSTEMS
26 FLOWCHARTS FOR THE SC/MP TEXT
27 ASSEMBLE YOUR OWN MI
28 USING MINI
29 A SIMPL

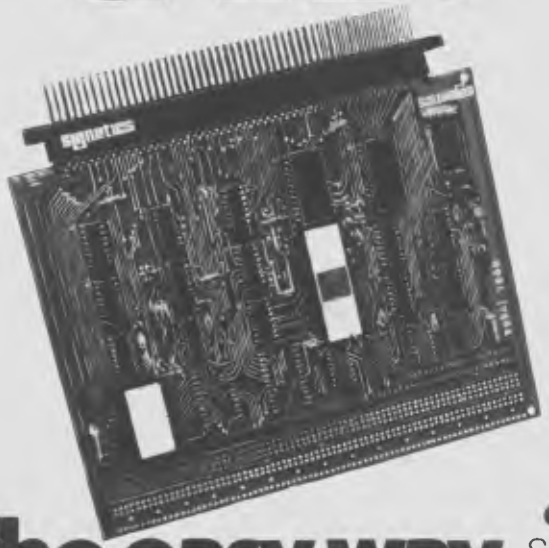
SUPPLEMENTAR

TWO PROGRAMS
MICROCOMPUTE
BOOKS ON MIC
SYDNEY & MELB



EDITED BY JAMIESON ROWE
AN ELECTRONICS AUSTRALIA PRODUCTION

ADAPTABLE BOARD COMPUTER PROTOTYPING CARD.



The easy way to start in Microprocessors

Two easy steps:
Plug in, Hook up
+ 5 volts, ground, and
a teletype,...and you're
in business!
Here is a short
list of features.

- 1 K BYTES ROM (PIPBUG EDITOR AND LOADER) BOARD EXPANDABLE TO 2 K BYTES ROM/PROM
- 512 BYTES RAM (2112B – 256 x 4 STATIC NMOS RAMS) BOARD EXPANDABLE TO 1 K BYTES OF RAM
- ON BOARD TTL CLOCK
- TWO – 8 BIT PARALLEL B1/D1 I/O PORTS
- RS232/TTY SERIAL I/O PORT
- CONNECTOR SUPPLIED
- AVAILABLE PRE-ASSEMBLED AND TESTED – 2650 PC1500 OR IN KIT FORM – 2650 KT9500

Ask your local Philips stockist to show you the 2650 PC1500 Adaptable Board Computer Prototyping Card, and get an easy start in Microprocessors.

PHILIPS ELECTRONIC COMPONENTS & MATERIALS,
P.O. Box 50, Lane Cove, 2066.
Sydney 427 0888,
Melbourne 699 0300, Brisbane 277 3332
Adelaide 223 4022, Perth 65 4199.

PHILIPS
Electronic
Components
and Materials

PHILIPS

153.0198

Getting into Microprocessors

Published by Electronics Australia

Edited by JAMIESON ROWE,

BA (Sydney), BSc (Technology, NSW), MIREE (Aust.)

Produced by GREG SWAIN,

BSc (Hons, Sydney)

First Edition, 1977

FIRST PRINTING — 1977

Printed by Magazine Printers Pty Ltd of Regent St, Sydney, and Masterprint Pty Ltd of Dubbo, NSW, for Sungravure Pty Ltd of Regent St, Sydney.

Contents

Main Chapters:

Supplementary Material:

Please note:

This book is based on articles that appeared in "Electronics Australia" from August 1976 to August 1977. As such, prices quoted for any of the equipment described herein should no longer be considered valid.

Getting into Microprocessors

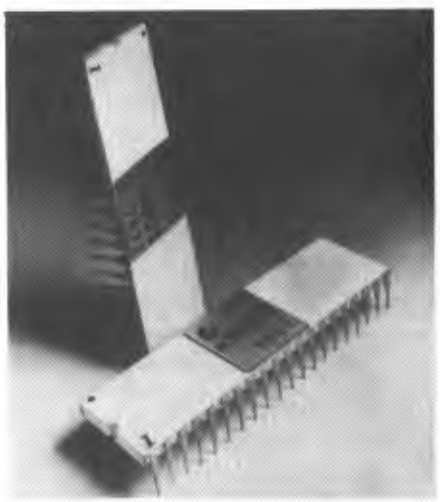
Five years ago, microprocessors and microcomputers were little more than dreams in the minds of research engineers and science fiction writers. Now they are a reality, and they seem set to change the whole face of electronics.

The change is happening already, and it is happening fast. So fast that overseas it is being described as an explosion, a revolution.

Many engineers and technicians in industry have been caught on the hop, and have found it difficult to adjust to the new devices. On the other hand many hobbyists have accepted the new devices with gusto, and some are pioneering all sorts of new applications. A healthy new market area has sprung up, to cater for the growing army of computer hobbyists.

Well, what are these new devices? Why are they having such a dramatic impact? Why are they having such an appeal for hobbyists? Why will most of us, whether professional or hobbyist, need to come to terms with them? How do you get started?

This supplement is the first of a number of features we are planning on this important subject, to answer questions like those above, and help you to "get into microprocessors". Editor Jim Rowe starts the ball rolling in the introduction which begins below, written to give you most of the background information you should know.



Most microprocessors come in 40-pin dual in-line packages, like the two shown here. These are samples of the Data General mN601 "microNOVA" chip, a 16-bit device which is software compatible with the same firm's minicomputers.

What exactly is a microprocessor?

Broadly speaking, a microprocessor is an integrated circuit (IC) which contains virtually all of the circuitry required to form the "heart" of a digital computer. Made using large-scale integration (LSI) techniques, it combines on a single IC chip the control, timing, data manipulation and arithmetic sections.

There are many different microprocessor ICs being made, and they vary quite widely in terms of potential computing capability. At the low end of the spectrum are relatively simple devices with a modest repertoire of different functions, and designed to deal with data in the form of 4-bit words. Rather more elaborate are the devices at the top end, generally with a much more powerful repertoire of functions, and designed to deal with 16-bit data words. In between these two extremes are many medium-level devices, most of them designed to deal with 8-bit data words or "bytes".

What is a microcomputer?

In general terms, as the name suggests, a microcomputer is simply a small computer—small in terms of physical size and cost, that is, not necessarily in terms of computing power. Nowadays there is also the broad implication that it uses a microprocessor IC as its functional heart.

Along with the microprocessor there is at the very least a memory of some sort, in which the "program" is stored. This is a sequence of binary numbers which the microprocessor interprets as "instructions", telling it how to perform the required tasks.

More specifically, there are two broad types of microcomputer. One type is similar to minicomputers and larger computers, in that it is designed to operate as a general-purpose computing system. The main memory is read-write or "random access" memory (RAM), and the computer may be made to perform virtually any required task simply by storing the appropriate program in the RAM. A series of different tasks may be performed one after the other, by feeding in and running a number of different programs in turn. This type of microcomputer therefore conforms quite closely to the conventional concept of a digital computer.

Microcomputers of this type are already tending to compete with established minicomputers, being capable of offering similar computing power at a fraction of the cost. It seems likely that in time, microcomputers will gradually eclipse both minicomputers and conventional large computers. Or putting it another way, all digital computers are likely to become microcomputers in the sense that they will be based on microprocessors and other LSI devices.

The other type of microcomputer still has a microprocessor controlled by a program stored in memory. But in this case the memory is a read-only memory or "ROM", and the program is therefore substantially permanent. The microprocessor and the ROM thus form a system which is designed to perform a single task, rather like a conventional custom-designed logic circuit.

This type of microcomputer is

described as being "dedicated", to emphasise the difference between its single-task behaviour and that of the general-purpose type. The task performed by a dedicated microcomputer may be quite complex, but it can only be changed by replacing the program in the ROM.

Just how this is done depends upon the type of ROM device being used. With mask-programmed ROMs, programmed by the manufacturer, it means replacing the ROM device as a whole with another. This must also be done with programmable ROMs (PROMs) of the type using fusible links. However, there are other PROMs which permit the stored program to be erased, either electrically or by exposure to ultra-violet light, and a new program stored in its place.

Why are these new devices having such a dramatic impact?

Basically there is a very simple and down-to-earth reason why microprocessors, in microcomputers, are forcing their way into our lives: cost. More and more, they are providing a way of doing things more cheaply than conventional ways.

By "things" we don't just mean here the traditional tasks done by computers—"number crunching", manipulating large slabs of data, etc. Microcomputers are certainly providing ways of doing this sort of thing more cheaply, but that's a relatively minor application. If this was the only application of microcomputers, they would scarcely be making a ripple.

In fact the horizon is very much wider than this. Microcomputers are capable of doing much more mundane tasks—the type of thing previously done by custom-designed logic circuits, relays, mechanical timers and sequencers, and so on. And they are fast becoming not only a more reliable way of doing these things, but far and away the cheapest way of doing them.

For example the latest automatic sewing machine to be released on the US market has a dedicated microcomputer inside, replacing the usual cams and other complex mechanics used to produce the various fancy stitches. Similarly the latest automatic washing machine to be released has a dedicated microcomputer replacing the traditional timing motor and sequence switches. And microcomputers have already started to appear in petrol pumps, traffic light controllers, process controllers, elevator systems, cash registers and other point-of-sale terminals.

In fact it is the dedicated type of microcomputer which seems to have the largest and most dramatic potential. We've probably only started to scratch the surface of its applications, because as yet we are still talking about existing jobs. Once we all get used to the idea of really low-cost "programmable black boxes", all sorts of applications are bound to open up—doing things which until now we wouldn't have even tried doing because they seemed too hard.

Why are microcomputers becoming the cheapest way of doing even quite mundane jobs?

There are two basic reasons why microcomputers are fast becoming the cheapest and most practical way of doing even mundane tasks. One is that IC manufacturing costs are lowered. Instead of having to produce many different types of specialised IC, the manufacturer is able to concentrate on only a few types: a microprocessor chip, RAM and ROM memory chips, and a few associated devices. These can be produced in really large quantities, giving economy of scale.

The other reason is that once a piece of equipment is designed using a microcomputer system, changes and modifications may be made far more cheaply. There are no costly changes to mechanical hardware, just changes to the "software": the program in the microcomputer memory. With dedicated systems, this is merely a matter of plugging in a new ROM with a modified program.

Why are engineers and technicians working in electronics finding it hard to adapt to microprocessors and microcomputers?

Microprocessors are providing problems for engineers and technicians because they are changing the whole approach to electronic equipment design. Traditionally, you designed a piece of equipment using conventional circuit design techniques—first developed in the days of thermionic valves, then adapted to transistors and ICs.

This involved selecting various specialised-function devices, and working out how they could be connected together to perform the specific job to be required.

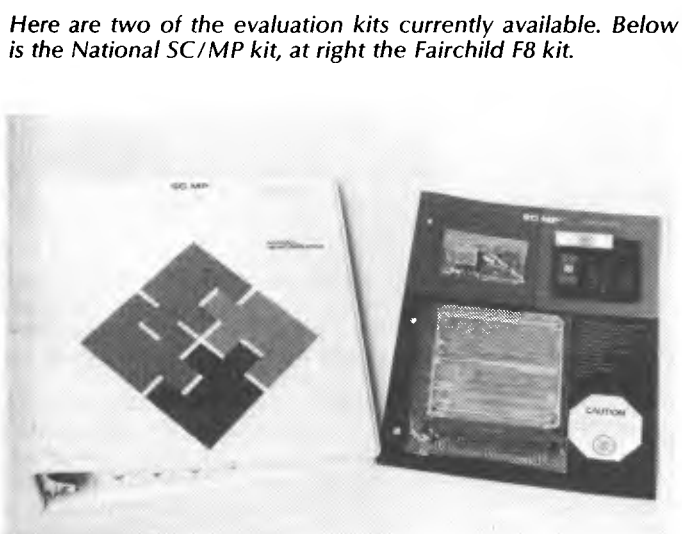
With microprocessors, this whole approach is becoming redundant. More and more, circuit design is becoming a matter of taking one "general-purpose black box" containing a microprocessor and a memory, and "telling it what to do". In other words, writing a program to put in the memory, so that the microprocessor is able to do the job.

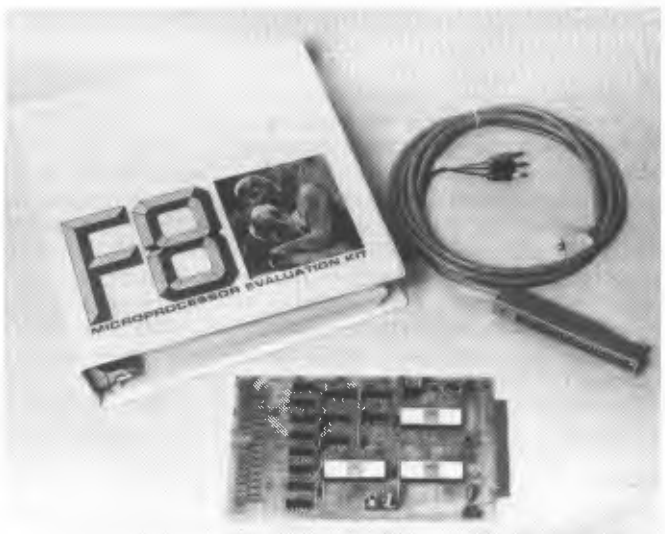
In short, circuit design is changing into computer programming, and circuit designers are finding that they must change into programmers. Much of their existing electronic design experience is becoming redundant, and they are having to learn a different set of skills.

How do you learn the skills required to work with microprocessors?

This depends to a certain extent on whether you are coming in at the professional level or at the hobby level, although the two are closer together than

Here are two of the evaluation kits currently available. Below is the National SC/MP kit, at right the Fairchild F8 kit.





one might perhaps think.

Not surprisingly, the firms making microprocessor ICs and the other devices involved have been motivated to help professional users adapt to the new devices. They have a vested interest in selling the devices, and this won't happen until engineers and technicians have learned what they are, what they can do and how to drive them.

Virtually all of the firms concerned have brought out great piles of literature supporting their chips: technical data, applications material, programming courses and so on. And many of them run intensive-course seminars designed to give the engineer or technician a fast down-to-earth introduction to their particular devices.

This is all very well, but there is a limit to what one can pick up from literature or from an intensive training seminar. To get real insight into microprocessors and microcomputers, there is no substitute for "hands-on" experience—actually sitting down and playing with one, writing and running small programs, and so on.

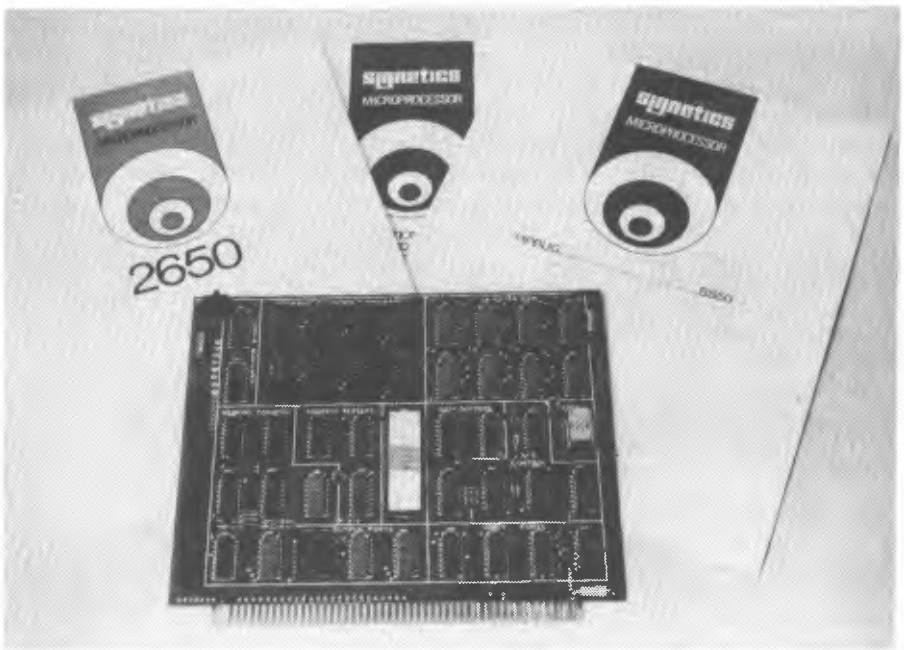
The companies soon realised this, and in an effort to meet the need they first came up with two rather different answers. One answer was to develop equipment to which they gave names like "prototyping development system". These were generally built around a microprocessor, but were basically a full-scale minicomputer designed to provide complete user support for writing, debugging and running programs intended for simpler systems. The cost was up in the many thousands of dollars.

The other answer was to make the same sort of facility available via the big computer time-sharing services, so that any firm with access to a time sharing terminal could do the same things.

In both cases, the user was presented with high-powered computing machinery "pretending" to be the sort of simple systems he was actually trying to design. And while this approach was fine for the very big user, who knew exactly where he was going and could afford to get there fast, it didn't really help those who weren't too sure what it was all about.

Happily in the last year or so, the manufacturers have been able to come up with a different answer, of much greater value to the small user and the person who still doesn't know if they are going to be a user or not. The exact format varies a bit, but broadly speaking this answer consists of a very basic microcomputer system built on a single PC board. It is usually called an "evaluation kit", and it is designed to provide the simplest, cheapest way of doing three things:

1. Getting "hands-on" experience with the microprocessor concerned.
2. Learning to write programs for it.
3. Developing programs for practical applications.



Here is the Signetics PC1001, larger of the two evaluation kits available based on that firm's 2650 8-bit microprocessor chip.

Most of the kits are designed to communicate with the user via an ASCII-code 110-baud teleprinter, such as the Teletype model ASR-33. However others include simple interfacing via a calculator-type keyboard and a set of 7-segment LED displays.

Exactly what do these kits provide?

Generally they provide a set of technical literature and user manuals, including a guide to programming. On the PC board, either already or when you have assembled it, there is the microprocessor chip, a ROM chip or chips, a RAM chip or chips, and a few other chips to provide a clock oscillator, a teleprinter interface and so on. More elaborate kits may have, or have provision for additional input-output interfacing, and perhaps provision for easy memory expansion, etc.

In the ROM already is a utility program described broadly as a "debug" program. Each firm tends to give their debug program a distinctive name, like "KITBUG", "PIPBUG", "FAIRBUG" and so on.

Debug programs differ a little, but generally what they do is allow the user to do the following things:

1. Feed a program into the RAM, either via the teleprinter keyboard, or via the punched paper tape reader if the teleprinter has one.
2. Examine any of the stored instructions, and modify them if required.
3. Run the user's RAM program, either all in one slab or in sections (the latter can help in finding why a program isn't working in the way you expected).
4. Examine any of the microprocessor's registers after the program has run, to see if all has gone as expected.
5. Punch out a paper tape version of the program via the teleprinter tape punch, if it has one.

These are just about all of the basic functions one needs to learn how to drive a microprocessor, and to develop small programs for practical applications.

The various functions listed above are provided by the debug program in response to simple commands typed in via the teleprinter. Each command has a code letter like "M" for examine memory, "P" for punch out the program, and so on, with the letter followed as required by numbers giving relevant address information—usually in hexadecimal code.

What do these evaluation kits cost?

It varies from firm to firm, and depends upon whether the kit concerned comes as an actual assemble-it-yourself assembly kit, or as a wired and tested board that only needs connection to a power supply and a teleprinter. Also some are more elaborate than others, to cater for those coming in at differnt levels of sophistication.

Currently prices vary between about $80 for a very simple kit, up to about $400 for a fairly pretentious one. These prices include full technical literature and user manuals, but don't include power supply or teleprinter, etc. ▶

Why have hobbyists taken so strongly to microprocessors and microcomputers?

Playing with computers can be great fun, as there is the same sort of intellectual stimulation provided by cryptic crosswords, puzzles, and games like chess. Once you've written even a simple program and finally got it going, the bug tends to bite. Before long, you're writing ever more adventurous programs, and hooking the computer up to all sorts of other equipment to have it perform ingenious tricks.

Of course before microprocessors came along, these delights were only available to a few lucky people. Mainly these were people who happened to work with computers or minicomputers, although there were a few hardy souls who built up their own small machines the hard way, using earlier ICs. Some of these people were those who built up the author's "EDUC-8" design, described in the issues of Electronics Australia from August 1974 to August 1975.

But what has happened within the last six months or so is that in coming up with the low-cost "evaluation kits" to help engineers and technicians get started with microprocessors, the manufacturers have at the same time produced what are in fact very tiny minicomputers. And this is the development which has triggered the dramatic increase in computer hobby activity.

To be sure, most of the evaluation kits are designed to go with ASCII-type teleprinters, and these are neither cheap nor readily available as far as the hobbyist is concerned. But there are ways around this problem. One way is to use an old Baudot-type teleprinter, and hook it up via a bidirectional code translator. Another is to build up a video terminal unit, based on an old TV receiver.

Here's where a magazine like Electronics Australia can help, by solving some of these practical interfacing problems. At E-A we are in fact already working along these lines, and we hope to publish details of interfacing units in the near future.

The main point to realise is that microprocessors and microcomputers are here, and that it is now possible for both professional and hobbyist to "get into them" with surprisingly little pain and financial strain.

Rockwell International make a large family of microprocessors and other associated chips, three of which are shown here.

To conclude this introductory look at microprocessors and microcomputers, here is a short and by no means complete survey of the evaluation kits currently available in Australia. Only brief details are given, and we hope to deal with many of the kits in more detail in forthcoming issues. For the present, however, the following may give you an idea of the type of system currently available. The kits are listed roughly in order of price.

1. **National SC/MP:** This is currently the lowest cost evaluation kit on the market, at $79.95 plus tax. It is based on the National Semiconductor SC/MP ("scamp") microprocessor chip, an 8-bit device designed primarily for low cost dedicated applications. The evaluation kit provides a complete basic system, with the KITBUG debug program in a 512-byte ROM together with a 256-byte RAM, a simple teleprinter interface and a crystal clock. It comes as an assemble-it-yourself kit, complete with full assembly instructions, user manual and programming manual. (NS Electronics Pty Ltd, cnr Stud Rd and Mountain Highway, Bayswater, Vic. Kits are available from all franchised distributors.)
2. **Fairchild F8 Kit:** This is based on Fairchild Semiconductor's F8 microprocessor, which is a two-chip 8-bit design. The kit comes as a wired and tested PC board, complete with an edge-connector socket already wired to a power supply/teleprinter cable; also a set of user, programming and applications manuals. The ROM has a capacity of 1024 bytes ("1k"), and comes with the FAIRBUG debug program in situ. The RAM also has a capacity of 1k, to allow development of quite elaborate user programs. Cost of the kit is quoted at $166.50 plus tax. (Fairchild Australia Pty Ltd, 37 Alexander St., Crows Nest, NSW 2065.)
3. **Mostek F8 Survival Kit :** Based on the F8 microprocessor which Mostek second-source from Fairchild, this kit comes in either assemble-it-yourself or fully assembled versions. The fully assembled kit provides 1k bytes of RAM, and 1k bytes of ROM with "DDT-1" debug program in situ. It also provides three 8-bit input/output ports, in addition to the teleprinter interface. The clock uses a quartz crystal. With the kit come a detailed application note, a programming guide and a listing of the DDT-1 debug program. Also an F8/ANSI Fortran Cross Assembler on punch cards,

to allow program development on large machines if desired. Approximate price of the D-I-Y kit is $158, with the assembled kit $200. (Namco Electronics, 239 Bay St., North Brighton, Victoria 3186. Also Total Electronics.)

4. **Signetics PC1500/KT9500 ABC prototyping system:** This is based on the Signetics 2650 microprocessor, an 8-bit device. The system comes either as an assemble-it-yourself kit (KT9500), or as an assembled PC board (PC1500). The system includes a 1k ROM with resident "PIPBUG" debug program, together with 512 bytes of RAM, two 8-bit latched input/output bidirectional ports, a crystal clock and a teleprinter interface. It also provides buffered data and address lines for subsequent memory expansion. The system comes with a 2650 technical manual, PC1500 applications booklet, PIPBUG listing and various technical notes. Price of the KT9500 kit is $165, with the PC1500 assembled system $245. (Philips Electronic Components and Materials, 67 Mars Road, Lane Cove, NSW 2066.)
5. **INTEL SDK80 System Design Kit:** Based on the Intel 8080 8-bit microprocessor chip, this comes as an assemble-it-yourself kit. On the PC board mount a 1k eraseable PROM with resident debug and monitor program, 256 bytes of RAM, a crystal clock, and three bidirectional 8-bit input/output ports as well as a teleprinter interface. There is also a second 1k eraseable PROM, for user program storage in addition to the RAM. The kit comes complete with assembly instruction manual, 8080 technical manual, and programming manuals. Price of the kit is $320 plus tax. (A. J. Ferguson Pty Ltd, 29 Devlin St, Ryde, NSW.)
6. **Signetics PC1001 Prototyping card:** This is based on the Signetics 2650 microprocessor, like the PC1500. The kit comes as an assembled PC board, with 1k of RAM and 1k of ROM containing the "PIPBUG" debug program. It has a crystal clock, and provides two 8-bit input ports and two 8-bit output ports as well as a teleprinter interface. The PC board also provides status indicator LEDs, and buffered address and data lines to simplify subsequent memory expansion. The kit comes with a 2650 technical manual, PC1001 applications booklet, PIPBUG listing, and various technical notes. Price of the PC1001 kit is currently $395. (Philips Electronic Components and Materials, 67 Mars Road, Lane Cove, NSW 2066.)
7. **National PACER System:** Developed by Hamilton/Avnet in the US, this is a packaged microcomputer system based on the National Semiconductor 16-bit PACE microprocessor chip. It comes as both a kit and an assembled unit, and has a case, complete with 8-character LED alphanumeric display panel, calculator-style keyboard and power supply. It has a 1k ROM with debug program, and 256 words of RAM. Further details are given in an article on PACER which appears later in this supplement. Price of the PACER kit is $595, with the assembled unit $695. (NS Electronics Pty Ltd, cnr Stud Rd and Mountain Highway, Bayswater, Victoria.)
8. **MICRONOVA 8562 microcomputer board:** This is a system produced by Data General, the minicomputer company. It uses the Data General mN601 16-bit microprocessor chip, with a powerful instruction set including hardware multiply and divide. The board includes 2k words of RAM, but a version of the system is available with 4k words (model 8563). The board includes buffering and timing circuitry for memory expansion to 32k words of dynamic RAM. Price of the 8562 board is quoted as $784. (Data General Australia Pty Ltd., 98 Camberwell Rd, Hawthorn East, Victoria.)
9. **OTHER MICROPROCESSOR CHIPS, ETC: A number of other microprocessor chips and associated devices are available from various firms, apart from those above. Some of these are listed briefly below.**

Motorola 6800 microprocessor family: From Motorola Semiconductor Products, Total Electronics, Cema Distributors.

General Instrument CP1600: From R & D Electronics.

MOS Technology 6502: From Digital Electronics (Marketing) Pty Ltd.

RCA COSMAC system: From Amalgamated Wireless Valve Co.

Rockwell PPS-4, PPS-8: From ANK Transmissions, Box A723 Sydney South, NSW 2000.

10. **OTHER MICROCOMPUTERS AND SYSTEMS: A variety of other microcomputer systems are available, apart from single-board evaluation kits as described above. Some of these are listed briefly below.**

MITS Altair 8800: Based on the Intel 8080 microprocessor, and designed for expansion to large minicomputer level. From WHK Electronic and Scientific Instrumentation, 2 Gum Rd, St. Albans, Victoria.

MITS Altair 680: Based on the Motorola 6800 microprocessor, with similar design approach to the 8800. From WHK Electronic and Scientific Instrumentation.

Intel 8080A Cramerkit: Based on the Intel 8080A microprocessor. From Ampec Engineering, P.O. Box 18, Strathfield, NSW.

Microprocessors: the basic concepts

If you've not had any experience to date with computers, a major problem in understanding and using microprocessors is likely to be the unfamiliar concepts involved. There is also a language problem, due to the use of many terms from computer technology. This introductory article has been written to help overcome both problems.

by FRED HORNE and BERNIE KUTE
National Semiconductor, Texas

Today, a computer connotes a machine that, once it is set up for a specific problem, performs a computation automatically and without human intervention. The present use of the term "computer" has a second connotation—it usually refers to an electronic machine, although mechanical and electromechanical computers do exist.

Two important factors dictate the intimate association between computers and electronics: no known principle other than electronics allows a machine to attain the speeds now commonplace in both large- and small-scale computers; and, no other principle permits comparable design convenience. In particular, digital computers use numbers that are respresented by the presence or absence of a voltage level or pulse on a given signal line. A single pulse defines one "bit" (short for binary digit, a base-2 number); a group of pulses considered as a unit is called a "word", where a word may represent a computational quantity or a machine directive.

For purposes of illustration, we shall compare two systems for solving simple mathematical expressions, both of which are comprised of the classical elements of a computer: an input/output device, a memory, a control section, and an arithmetic and logic unit or ALU (the computational element). The control section, together with the ALU, is considered to be the central processing unit (CPU). (See Fig. 1.)

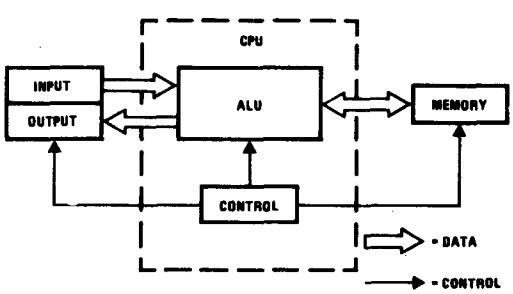


FIGURE 1. Basic Elements of a Digital Computer

The first system (Fig. 2) is comprised of a man and a calculator. The man's fingers represent the input, his eyes coupled with the calculator's output represent the system output, the calculator electronics function as the ALU, and his brain serves as the memory as well as the control section. Here is the sequence of events that occurs when our man-calculator solves the problem 6 + 2 = ?

1. Brain accesses first number to be added, a "6";
2. Brain orders hand to depress "6" key;
3. Brain identifies addition operation;
4. Brain orders hand to depress "+" key;
5. Brain accesses second number to be added, a "2";

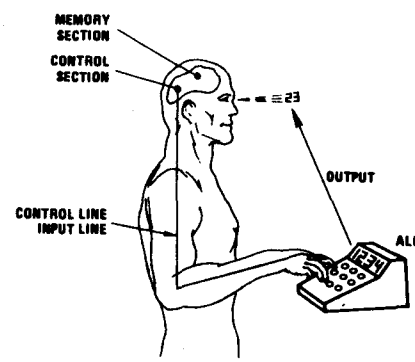


FIGURE 2. Man + Calculator = Computer

6. Brain determines that all necessary information has been provided and signals the ALU to complete computation by ordering hand to depress "=" key;
7. ALU (calculator) makes computation;
8. ALU displays result on readout;
9. Eyes signal brain, brain recognizes this number as the result of the specific calculation;
10. Brain stores result, "8", in a location that it appropriately identifies to itself to facilitate later recall.

We shall now develop a classical computer and illustrate how it might be used to solve the same problem. To begin, note that the memory (Fig.3) is composed of storage space for a large number of words: each storage space is identified by a unique "address". The word stored at a given address may be either computational data or a machine directive (such as add, read from memory, etc.).

Two temporary storage registers, each capable of containing one word, complete the memory. These registers are designated as "memory address register" (MAR) and "memory data register" (MDR). The MAR contains the binary representation of the address at which information is to be read out of memory or written (stored) into memory, while the MDR contains the data being exchanged with memory.

Turning to the ALU, (Fig. 4) shows that

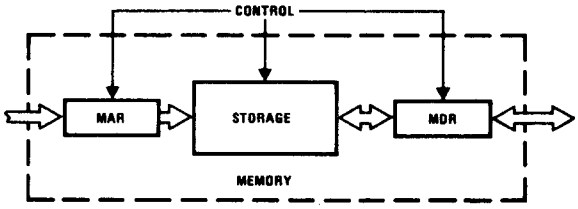


FIGURE 3. Elements of a Memory

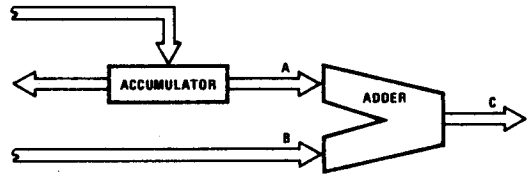


FIGURE 4. Arithmetic and Logic Unit

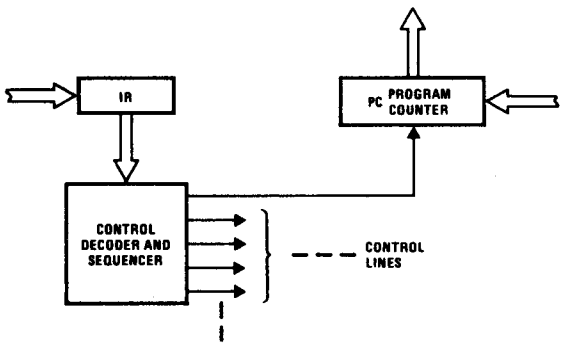


FIGURE 5. Computer Control

FIGURE 6. I/O Register Interface

TABLE 1. Sample Program

Memory Location	Instruction (Contents)
100	Input to accumulator
101	Store accumulator at 50
102	Input to accumulator
103	Add accum, Loc. 50
	Place result in accumulator
104	Store accumulator at 60
105	Halt

this portion of a computer, in its simplest form, comprises an "adder" that adds (or performs similar logical operations upon) two inputs A and B and produces an output at C, and an "accumulator", which maintains intermediate results of a computation or numbers for a pending computation.

The remainder of the CPU, the control portion, is implemented using an "instruction register" (IR), a "control decoder and sequencer", and a program counter (PC). These are shown in Fig. 5. A machine directive (instruction) is transferred into the IR and is subsequently interpreted by the decoder/sequencer, which issues the appropriate control pulses to the other computer elements.

The PC contains, at any given time, the address in memory of the next machine directive or instruction. This counter is normally incremented by a count of one immediately following the reading of a new instruction. The PC contents may be replaced by the contents of a specified memory location if the last instruction was of the "jump" class. This causes the next instruction to be read from a program-specified location, instead of from the next sequential location as is the general rule.

Finally, a means of input/output (I/O) is provided by an "I/O Register", through which data is exchanged with external (peripheral) devices (Fig. 6).

We have now collected all the basic elements of a computer; all that remains to do is to interconnect them into a functioning, automatic processor. Fig. 7 shows such an interconnection, and represents a complete computer.

The analysis continues with the execution of the same problem used to illustrate the man-calculator, but somewhat rephrased:

"Read-in a number from the I/O. Store it in memory location 50. Read-in another number from the I/O. Add the two numbers together. Store the result in memory location 60, and halt."

A "program" has been written to execute this task, and is stored in consecutive memory locations beginning at 100. This program, written in an artificial symbolic language, is shown in Table 1.

All computers spend about equal periods of time in one of two distinct states: "fetch", or "execute". In the fetch state, the computer reads from memory the next sequential instruction and places it in the instruction register (IR). In the execute state, that instruction is carried out as a series of transfers from one register to another and as various ALU operations. Table 2 examines the program shown in Table 1, as it is actually execu-

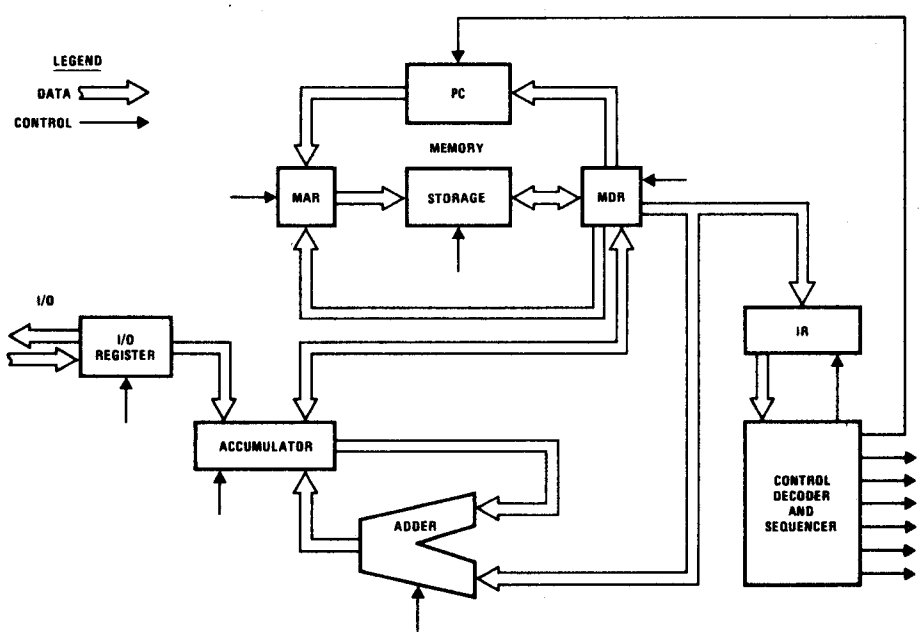


FIGURE 7. Simplified CPU and Memory

ted, by specifying the contents of each register at each machine cycle (time interval) and assuming the computer is now ready to fetch the first instruction in our program.

All computers (processors, CPU's, etc.) operate in a similar manner, regardless of their size or intended purpose, although many variations are possible within the basic architectural framework. Common variations include, for example, highly sophisticated I/O structures (some of which have direct and/or autonomous communications with memory), multiple accumulators for programming flexibility, index registers that allow a memory address to be modified by a computed value, multilevel interrupt capability, and on and on.

One of the most exciting architectural concepts to gain popularity in the past few years is that of microprogrammed control. A microprogrammed computer differs from the classical example in its control-unit implementation. The classical machine has for its control unit an assemblage of logic elements (gates, counters, flip-flops, etc.) interconnected to realize certain combinatorial and sequential Boolean logic equations. On the other hand, a microprogrammed machine uses the concept of a "computer within a computer". That is, the control unit has all the functional elements that comprise a classical computer, including read-only memory (ROM).

The "inner computer", which (generally) is not apparent to the user, executes the user's program instructions by executing a series of its own microinstructions, thereby controlling data transfers and all functions from computed results. And this means that changing the stored microprogram that generates the control signals alters the entire complexion of the computer. By altering a few words stored in the ROM, the computer behaves in an entirely new fashion—it can execute a completely different set of instructions, simulate other computers, tailor itself to a specified application. It is this capability for "custom-tailoring" that allows a microprogrammed machine to be optimized for a given usage. By so extracting the utmost measure of efficiency, a microprogram-controlled machine is less costly and easier to adapt to any given situation, no matter how diverse or demanding.

It is possible to program a device that isn't a computer at all. An operational amplifier, for example, is a circuit that is basically a multiplier. Something is put in, something comes out; the op amp performs a linear function. But this building block can do something other than multiplication: a capacitor, for example, connected from the op amp's output to its input, creates a "programmed-by-wire" integrator.

As it is with the op amp, so it is with the microprocessor. A microprocessor is a super circuit—a black box with a transfer function that changes in accordance with a set of commands called a program. Inside the black box (i.e., on the chip) is a collection of building-block logic—an assemblage of many logic elements. You can in fact replace the microprocessor in any system with sets of random logic on PC boards, but you would have to change the logic boards on each clock pulse!

Thus, if you know what a flip-flop does you know what it does inside or outside a microprocessor; an AND gate AND's whether it's inside a microprocessor or on a lab bench. But in a microprocessor literally thousands of such logic elements are squeezed onto one or two chips. And this creates a problem: too much information, too few pins.

To overcome the pin problem, microprocessor manufacturers strap every logic element to every other logic element through a set of buses that allows mutual, element-to-element communications. Bus connections are made through a series of electronic switches; opening and closing the switches transfers the data through the microprocessor's maze to produce a control function. And it is software that sets the switches. System software is a set of tools, supplied by the microprocessor manufacturer, that allows you to construct application programs—programs that let the microprocessor do something.

To appreciate what software does for you, consider an elementary operation such as addition. Get A, get B, add them together and come out with C. Easy? In decimal notation, yes. But this trivial problem is not quite as simple when one speaks in binary. Dealing with long binary numbers is complex and difficult because one's and zero's aren't a natural language for Homo Sapiens. We have problems trying to figure out what's going on when we look at raw binary; writing it is even more troublesome.

Can you imagine looking down 14 sheets of printout, each with 65 lines of binary gibberish, attempting to determine what you did wrong? Yet this is ultimately how you program a job on a microprocessor. You have to write the story of how the processor is to wire itself from microsecond to microsecond. So all system software, the whole range of it that every manufacturer offers, is aimed at only one thing: to get you from the stated idea to the working program as painlessly and as rapidly as possible.

In the construction of application software, you first evolve a flowchart (Fig. 8A) that describes the functions to be performed and their order. (At this stage your thought processes and activities resemble those of the random-logic designer.) Once the chart is laid out, you start to code the program in either a high-level or a mnemonic-shorthand language that both you and your system under-

TABLE 2. Register Content

NOTES	PC	ACCUM.	MAR	MDR	I/O REG.	IR	MEMORY (R=READ) (W=WRITE)	STATE
	100	?	?	?	?	?	?	?
Start	100	?	100	(100)	?	(100)	R	Fetch
Input	100	6	100	(100)	6	(100)		Execute
	101	6	101	(101)	?	(101)	R	Fetch
Store	101	6	50	6	?	(101)	W	Execute
	102	6	102	(102)	?	(102)	R	Fetch
Input	102	2	102	(102)	2	(102)		Execute
	103	2	103	(103)	?	(103)	R	Fetch
	103	2	50	6	?	(103)	R	Fetch
Add	103	8	50	6	?	(103)		Execute
	104	8	104	(104)	?	(104)	R	Fetch
Store	104	8	60	8	?	(104)	W	Execute
	105	8	105	(105)	?	(105)	R	Fetch
Halt	105	8	105	(105)	?	(105)		Execute

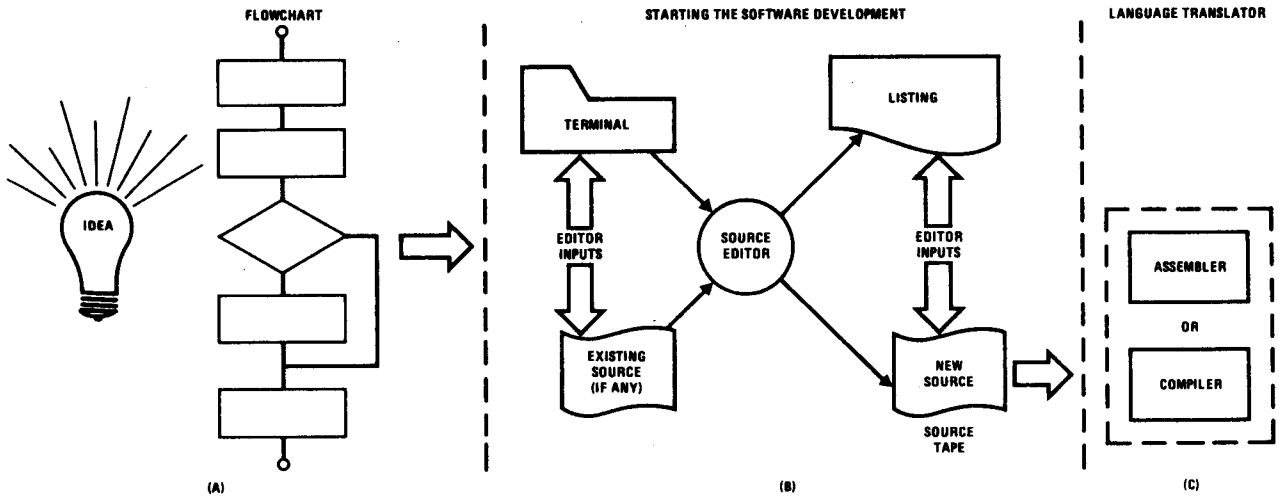


FIGURE 8. The programmer's ideas, expressed in a flowchart, are written out in mnemonic form to serve as Editor inputs. New inputs plus sections of existing programs are combined to form a new Source; this Source is the input to the language Translator.

stand. Here you encounter your first piece of software, the Text or Source Editor program (Fig. 8B).

Most microprocessor users write on continuous media (paper tape or cassettes), which do not allow you to get in and pull out one piece. Thus, corrections on a continuous source involve making a wholly new source—a constant problem and an awfully wasteful task. But there is a utility program called a Source Editor that lets you do the entire job with a teleprinter and a microprocessor Development System. If you make an error, just tell the Editor what changes to make and it's done! The Editor helps you "massage" the source code until it looks like it's going to work. Then, with the corrected (?) program in the Editor's memory cells, you push a button and a paper tape (or whatever) is put into your hands.

The "whatever" that has just been put into your hands has one minor, relatively insignificant, but fatal error—the microprocessor cannot understand a single bit or byte of it. But do not despair: an electronic Translator program (Fig. 8C) converts the continuous, source-mnemonic shorthand into something the microprocessor can understand.

The Translator (Fig. 9A) takes the source tape and generally gives back three outputs:

- The Program Listing—a copy of both the source and binary object codes;
- The Error Listing—a roster of all grammatical, label, and syntax errors; and,
- The Binary Object Code—a paper tape (or whatever) with the machine-readable binary translation of the program.

But there are two types of Translators—the Assembler and its exotic cousin, the Compiler—and there may be some argument as to which translation device is the more useful: Should you use an Assembler or a Compiler to translate the mnemonic sources? The difference is in the mnemonics.

If you happen to have run programs on minicomputers, then you've been exposed to the so-called "assembly language" mnemonics: LD means load; JMP means jump; ST means store; etc. It's the shortest language (outside of raw binary) used to talk with the processor. Programming with this shorthand is a bit tricky but an assembler-type Translator gives you a better feel for the machine and you can usually pare down the number of statements necessary to get the message across; and this saves time and money.

On the other hand, a compiler-type Translator lets you write in a high-level language that looks like English (Fortran, etc.). Its statements can easily be read by someone with no training at all. The Compiler translates these statements into a series of machine commands that carry out the desired function with the advantages of faster programming and a self-documenting program that you can read directly. But you often pay for this ease of use: since the Compiler deals with more general statements, it often translates in an inefficient way using more machine commands than really necessary at that level. Extra statements consume memory and result in slower program execution.

So, in retrospect, Compilers cut programming time and costs, but raise system costs. Assemblers do just the opposite. Which should you use? Compilers are most useful to those of you who constantly re-program your systems and make few versions of each program. Assembler users, on the other hand, will be those of you who will program the system once, then reproduce it a thousand or more times; programming costs are amortized over the production run and in memory savings.

At this point in the writing of a program many of you will wish that you could forget the whole thing, for there are programs with one hundred code lines that come out of the Translator with four hundred errors! But forge onward. Make another pass through the Source Editor (and another, and another...), to correct the errors that the Translator has spotted.

Eventually, you will get your reward, the sweetest line ever printed on a computer listing: "ASSEMBLY COMPLETE —NO ERRORS."

Actually, that statement simply means that the Assembler didn't find any errors. And you soon find out that this has almost nothing to do with whether or not the program will run on a machine. The reason is that the Assembler, although it helps you weed out logic errors from the program that you wrote, cannot tell you whether or not that program does exactly what you think it's going to do. In other words, there can be (and very probably will be) logic differences between your vision of what's needed to perform a function and that of the machine. Such an error may be one as simple as your forgetting to set a flag at some point; unimportant, perhaps, to your charting of a problem's solution, but all-important to the machine for without that bit of information your program cannot run. But other utility programs (such as DEBUG) are available to help you solve such problems.

Now that the Translator has provided you a binary tape with your program on it, you must somehow get the program into the machine's memory along with whatever other software routines your program needs for operation. The Loader program (Fig. 9B) does this for you;

it reads your tape into a microprocessor Development System (Fig. 9C), allocates memory space to the program, and stores the code in the appropriate location. Typically, several sections of memory are needed for different functions (executable code, interrupt calls, subroutine linkages, etc.), and it is up to the Loader to see that each part of the program is put into the right place. Loaders are available to load from teleprinters, paper-tape readers, and, sometimes, high-speed bulk storage devices.

Once the program is loaded, you cross your fingers and hit the RUN switch. As we've already said, very probably nothing will happen.

Now, if you are using random logic and find it doesn't work, you unplug it, repair any damaged hardware, and then try to determine what's wrong. With an oscilloscope on the gates and clocks, you try to see what's happening. But in the microprocessor only one set of logic exists, re-wiring itself at the speed of light. If you don't have any idea what's going on, the oscilloscope can't help you. What you need is a different type of fault-finding tool.

The tool is a program, called DEBUG, that lets you use a Teletype as a scope to help you find out what's happening. DEBUG is loaded into a Development System first, then your program is entered. You peck away at the TTY and say, "DEBUG, run my program from here to there, stop it, and tell me what is in memory". The TTY rattles and you've got the answer on a printout. "Show me what is in these accumulators." DEBUG does! "Show me this, show me that." Done, done.

As your program is stepped through, you'll encounter parts that don't work. These snags are cajoled and fondled individually until the whole thing runs—perfectly—and you have a working object code that represents your algorithm in ones and zeros.

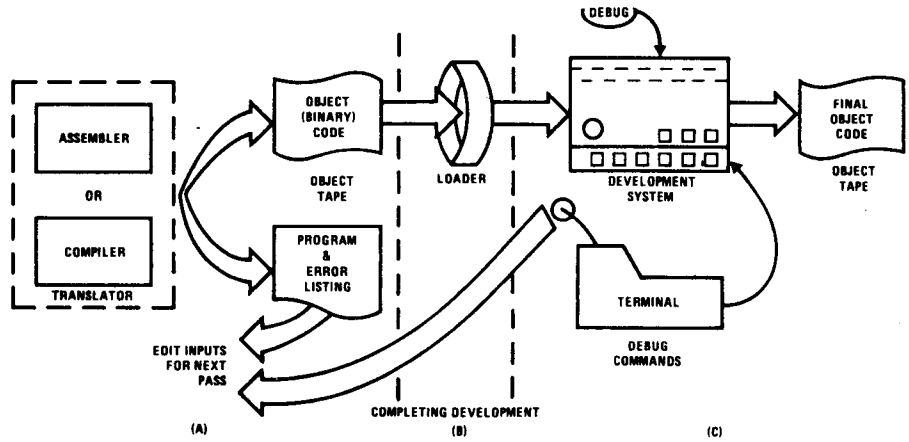


FIGURE 9. Translator outputs include: an Error Listing (to serve as Editor inputs on the next pass); a Program Listing; and a tape of the translated program (the Object Tape). The Object Tape is deposited by the Loader into Read/Write Memory inside the Development System. Here the new code is run by the DEBUG program according to commands input by you. The code can be modified via terminal inputs until it runs properly; working code is then dumped from memory. Note that although a workable object tape may exist at this time, your job is not complete until you edit and retranslate your source to produce code identical to the working code.

There is an alternative to the microprocessor debug section of a Development System. It is called a Simulator, and it typically runs on a large computer and includes both debug and simulation. To use it, load the binary code into the computer, call the Simulator, and then direct it to exercise the code to find the defects. However, this approach can only take you part of the way; it will not isolate timing problems that have to do with the outside world.

When the Simulator wants an input, it stops and asks for one. You sit there and peck away at the typewriter; which is fine if you want to test things that are slow. But if you wish to test a program that operates, say, a 100kHz 1/O converter, you won't be able to keep up with it. So the Simulator can only take you so far. Ultimately you have to return to the hardware prototype approach, and this is why the microprocessor manufacturers have felt it necessary to produce sophisticated hardware prototyping tools.

We at National believe a Simulator really doesn't help. We encourage users to take the Development System itself, put in the actual interfaces to be used, and use DEBUG to massage the program in real-time and watch what it does.

(Reprinted from the National Semiconductor publication "Logic Designer's Guide to Program Equivalents to TTL Functions", Copyright 1976, by arrangement).

MICROPROCESSOR JARGON: the words you need to know

ACCUMULATOR: Specifically, a data storage device (register) for work in progress; part of the equipment in the arithmetic unit of a processor, in which arithmetical and logical operations are performed (the ALU).

ADDRESS: A number that designates a register, a memory location, or a device.

ADDRESS FIELD: That part of an instruction or word containing an address or operand.

ASSEMBLER: A program that translates symbolic language to machine language.

BINARY: Involving a choice or condition of two alternatives (yes/no; on/off); a number system using the base 2.

BIT: Binary digit.

BUFFER: An area of memory that is used as a work area or to store data for an input/output operation.

BUS: A circuit over which data or power is transmitted.

BYTE: A group of consecutive binary digits usually operated upon as a unit.

CARRY: A condition occurring during addition when the sum of two digits equals or exceeds the number base; or, the digit to be added to the next higher column as a result of the sum overflow.

CENTRAL PROCESSING UNIT (CPU): The portion of any computer that consists of the arithmetic unit, the control unit, and the storage unit.

CLOCK: A master timing device used to provide the basic sequence pulses for the operation of a synchronous computer.

COMPILER: A program that produces a machine-language program from a source-language program.

COMPLEMENT: In the binary number system there are two complements: the "ones complement," and the "twos complement". The ones complement is obtained by converting all ones to zeros, and all zeros to ones. The twos complement may be obtained by first converting a binary num-

Continued next page

Microprocessor jargon ctd. . . .

ber to its ones-complement and then adding one to the ones-complement. In binary logic, signals may be in one of two possible states: *true* or *false, high* or *low, on* or *off.* Thus, a signal is complemented by changing it from one state to the other state.

CONDITIONAL BRANCH: A branch that occurs only if a certain condition is present in the machine at the time the instruction is executed.

CONSOLE: The portion of the processor that may be used to control the machine manually, correct errors, determine the status of registers, counters, and storage, and manually revise the contents of storage.

CONTROL SECTION: The part of a processor that determines the interpretation and execution of instructions in their proper sequence, including the decoding of each instruction and the application of the proper signals to the registers, arithmetic and logic units in accordance with the decoded information.

DATA: A general term loosely used to denote any or all facts, numbers, letters, and symbols that can be processed or produced by a processor.

DEBUG: To isolate and remove malfunctions from a computer or mistakes from a program; also, a utilities program that helps correct application programs.

DIAGNOSTIC ROUTINE: A specific routine designed to locate either a malfunction in the processor or a mistake in coding.

EFFECTIVE ADDRESS: The addition of the contents of the base register and displacement plus, in some cases, the index register contents to form the address actually used in addressing main memory.

ENABLE: Restoration of a suppressed interrupt.

EXECUTE: To carry out an instruction or perform a routine.

FLAG: A bit used to indicate the status of an element.

FETCH: To retrieve a word of data from main memory.

FIRMWARE: Read-only memory (ROM), or the data or instructions stored in ROM.

HALT: A machine instruction that stops the execution of a program.

HEXADECIMAL: Related to a number system that uses the base 16.

HARDWARE: The physical equipment of the processor.

INDEX REGISTER: A register that modifies the operand address in an instruction or base address to yield a new effective address.

INITIALIZE: A program or hardware circuit that clears registers and sets counters and switches to their starting values.

INSTRUCTION: A user-coded macroinstruction that causes the microinstructions to perform certain operations.

INTERRUPT: A break in the normal flow of a system such that the flow can be resumed from that point at a later time. An interrupt is usually caused by a signal from an external source.

JUMP: An instruction or signal that, conditionally or unconditionally, specifies the location of the next instruction and directs the processor to that instruction.

LABEL: An ordered set of characters used to symbolically identify an instruction, an address, or a value.

LIST: An ordered set of items.

MACHINE LANGUAGE: The system of (binary) codes by which instructions and data are represented internally within a data processing system.

MACROINSTRUCTION: In general, any single instruction that causes a complete sequence of events to occur; a single instruction made up of a number of microinstructions that together perform a specific operation. A microinstruction is carried out in one microcycle.

MAIN MEMORY: Read/write memory that is external to the control ROM but is internal to the microprocessor.

MICROCYCLE: The basic machine cycle of the microprocessor.

MICROCODE: The steps or microinstructions of a microprogram, or the binary coded data contained in the microinstruction words of the control ROM.

MICROINSTRUCTION: See MACROINSTRUCTION.

MICROPROGRAM: A set of basic instructions (microinstructions) stored in read-only memory, programmable read-only memory, or read/write memory, and used by the control section of a processor to command registers, arithmetic and logic units.

MICROPROGRAMMING: Machine-language coding in which the coder builds his own machine instruction from the primitive basic instructions built into the hardware.

MNEMONICS: Operation codes written in easily-remembered symbolic code rather than the actual machine code.

OPERANDS: Any quantities entering or arising in an operation. An operand may be an argument, a result, a parameter, or an indication of the location of the next instruction.

OVERFLOW: The condition that arises, in a digital computer, when the result of an arithmetic operation exceeds the capacity of the storage space allotted.

PROGRAM: A group of related routines that solve a given problem.

PROGRAM COUNTER: A counter constructed in hardware that contains the address of the next instruction to be executed.

READ-ONLY MEMORY (ROM): A hardware (semiconductor) data storage device that may be programmed similar to read/write memory but that cannot be erased without destroying the device; the stored data may be read, but not changed.

READ/WRITE MEMORY: A hardware (semiconductor) data storage device in which the stored data may be read as well as changed; common usage refers to R/W memories as random-access memories (RAMs).

REAL-TIME: The performance of a computation during the actual time that the related physical process transpires.

REGISTER: A hardware device used to store a computer word, where the word is to be manipulated as either data or an instruction.

ROUTINE: A set of coded instructions arranged in proper sequence to direct the processor to perform a desired operation or series of operations.

SIGN BIT: The bit position in a computer used to designate the algebraic sign of the word.

SHIFT: To move an ordered set of bits one or more places to the right or left.

SOFTWARE: The totality of programs and routines used to extend the capabilities of computers (such as compilers, assemblers, routines, and subroutines).

SOURCE LANGUAGE: The high-level (often mnemonic) language in which you specify a program for the computer. It is translated (by Assembler or Compiler programs) to a machine-readable binary code.

STORAGE: Any device into which units of information can be copied.

SUBROUTINE: A series of computer instructions that performs a specific task for many other routines.

WORD: An ordered set of characters that occupies a single storage location and is treated by the computer circuits as a unit and transferred as such.

WRITE: To transfer information to a device.

The Intel 8080A

Without a doubt the most popular and widely used of all current microprocessors is the Intel 8080A, the first of the "second generation" devices and one which is now supported by a very broad range of hardware and software. This article looks at the 8080A and its companion chips and then reviews the Intel SDK-80 evaluation and system development kit.

by JAMIESON ROWE

The Intel Corporation was the first to develop and market a microprocessor, back in 1971. In fact Intel's founder and Chairman Dr. Robert Noyce is acknowledged as the inventor of the microprocessor, and apparently still holds key patents.

The first Intel microprocessor was a 4-bit P-channel device, the 4004, with 46 instructions. This was followed by an enhanced 4-bit device with a repertoire of 60 instructions, the 4040. Then in 1973 came the first P-channel 8-bit processor, the 8008. This had a repertoire of 48 instructions, and an instruction cycle time of 20us (now 12.5us).

But undoubtedly the most popular Intel microprocessor to date has been the 8080, an enhancement of the 8008 which came on the market in early 1974. An N-channel device, the 8080 is about 10 times faster than its predecessor. It also has a larger instruction repertoire of 78 instructions.

The 8080 has been the most popular microprocessor ever produced, and even though it may lack some of the features offered by later devices, it seems likely to remain popular for many years.

Intel themselves have announced a number of newer microprocessors, including a more self-contained device called the 8085 and a one-chip microcomputer with internal ROM and RAM called the 8048. But they are at pains to point out that these newer chips are not regarded as superseding the 8080.

One of the prime attractions of the 8080 from the commercial point of view is that it is now supported by a tremendous amount of software and applications information—much of it user generated. This inevitably tends to reduce development costs of an 8080 system compared with a system based on one of the newer chips, if other considerations are equal.

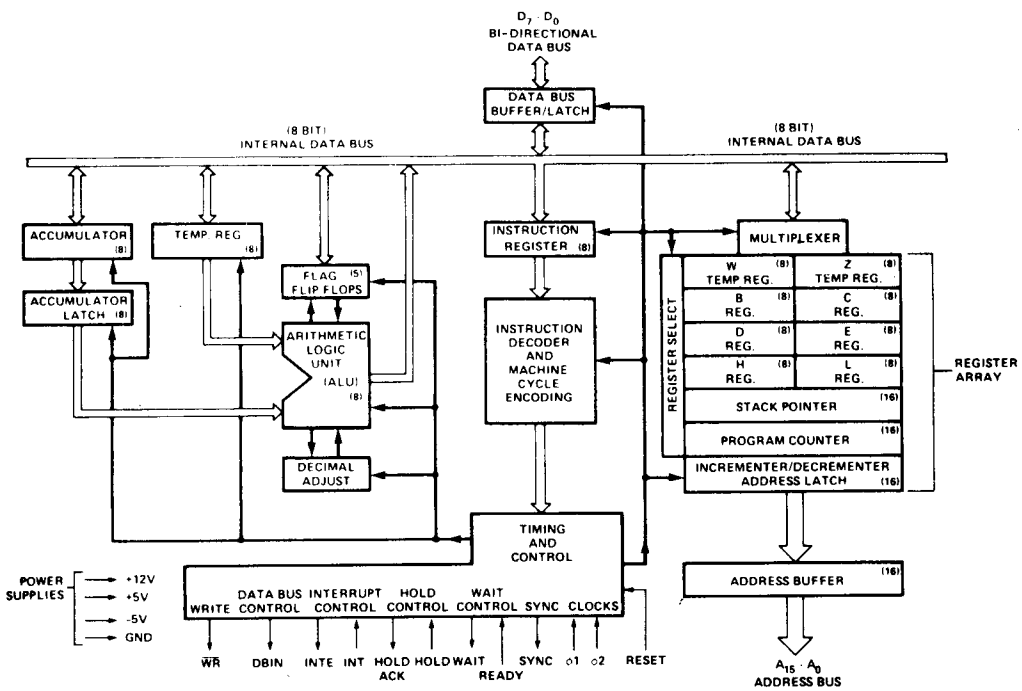
The block diagram for the 8080 CPU chip is shown below. Its architecture is not unlike many minicomputers. There are 10 internal registers accessible to the programmer, two of which are 16 bits long: the program counter and a stack pointer. A third register called the Program Status Word (PSW) provides the five status flags.

The remaining 7 registers are all 8 bits long. One is the single primary accumulator A, while the other six are secondary accumulator/data counters. Two of these, designated the H and L registers, are generally used together as a 16-bit data counter. The other four registers can also be grouped together in this way for some instructions.

Data and address information pass between the 8080 and the rest of a system via two separate buses, an 8-bit bidirectional data bus and a 16-bit address bus. The latter gives the 8080 the ability to directly address 65,536 or "65k" bytes of memory. The output buffers on both the data and address bus lines are 3-state, and may be disabled for external control of the system bus. This facilitates DMA operation, for example.

The 8080 implements its stack in external RAM, and as the stack pointer register is 16 bits long this means that the stack may be virtually anywhere in the 65k of memory space.

The 8080 status register or PSW has five flag bits. These signify zero result, result sign, carry, even parity, and 4 bit carry (for BCD arithmetic). None of the status



A functional block diagram for the 8080A microprocessor itself. There are 10 internal registers accessible to the programmer, two of which are 16 bits long: the program counter and the stack pointer. Apart from the 8-bit primary accumulator A there are six secondary accumulator/data counter registers and a 5-bit status register.

flags is accessible directly via external device pins, only internally via certain instructions.

Although I/O devices may be interfaced so that they appear in 8080 memory space, and are accessed via memory address instructions, the 8080 also has facilities for separate I/O addressing. 8 bits are allocated for peripheral addresses, so that there is potential for addressing up to 256 input ports and 256 output ports quite separately from normal memory space.

The 8080 chip uses N-channel silicon gate MOS technology. It requires three operating voltages: +5V, −5V and +12V.

There is no clock oscillator on the 8080 chip itself, which requires high level non-overlapping two phase clock signals. These are normally provided by the 8224, a companion device designed specifically as a clock generator and driver. It also provides synchronisation for certain system control signals.

Some of the system control signals generated by the 8080 are multiplexed out on the data bus lines, during one of the clock phases. To provide demultiplexing of these signals most 8080 systems use another companion device, the 8228 System Controller.

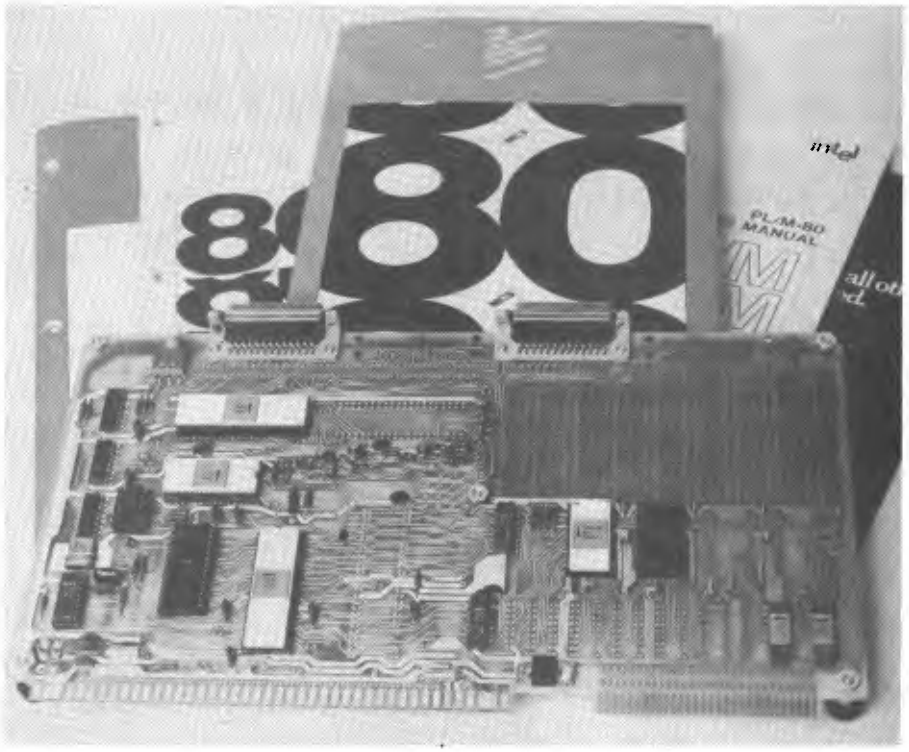
Actually the 8228 is not only a control signal demultiplexer, but a transceiver/driver for the data bus as well. And it also provides automatic control of 8080 interrupt vectoring—providing a hardware generated vector itself when none is provided by the interrupt source.

In responding to an interrupt request, the 8080 looks for an externally-supplied vectoring instruction. In most cases this is a one-byte jump to subroutine or RST instruction, capable of specifying one of eight subroutine starting addresses in the first 63 bytes of memory space. However for systems required to service more than eight interrupt sources, it is possible to use a 3-byte CALL instruction to routines anywhere in memory space.

Apart from the 8224 and 8228, Intel has provided the 8080 with a number of support devices to facilitate system design. Among these devices are the 8212 8-bit I/O port, the 8255 Programmable Peripheral Interface—which provides 24 bits of bidirectional parallel interfacing—and the 8251 Programmable Communication Interface which is basically a USART with programmable formatting and control.

There is also a large and still growing family of memory devices, with ROMs and PROMs as well as RAMs. These include a 2k byte mask-programmed ROM and a 1k byte UV-erasable PROM, with larger devices coming.

Also coming are more specialised interfacing devices, such as a programmable DMA controller, a programmable interrupt controller and a floppy disk controller and formatter.



The smallest 8080A-based system available from Intel themselves is the SDK-80 assemble-it-yourself kit, shown here assembled and with its accompanying manuals. A review of the kit is given in this article.

The instruction set of the 8080 includes 1, 2 and 3-byte instructions. While it is nominally said to comprise 78 different instructions, in fact almost all of the 256 possible op codes are used for distinct instruction variants.

Thus the data move or MOV instruction actually has 63 different variants, with different sources and destinations specified—including 14 which involve implied memory addressing. Similarly each of the 8 accumulator operate instructions has 8 variants, while the increment and decrement instructions have 12 variants each.

Included in the 8080 repertoire are 10 jump instructions, 9 CALL and 8 RST instructions for subroutine calling, 9 subroutine return instructions, 10 instructions for operating on the stack, and 20 immediate-addressing data instructions —four of which handle 16-bit data.

Apart from immediate addressing, the 8080 provides only two memory addressing modes. One is direct absolute addressing, used for 3-byte load, store, jump and call instructions. The other mode is implied addressing, used for all other memory reference instructions. These are all 1-byte instructions, and generally use the H and L registers as a 16-bit data counter.

An interesting aspect of the 3-byte instructions using direct absolute addressing is that the second instruction byte carries the eight LEAST significant address bits, with the third byte having the eight MOST significant bits. This is the opposite of the scheme used by virtually all other microprocessors, and slightly confusing at first. However, it doesn't take long to make the mental adjustment.

The variety of conditional branching instructions provided by the 8080 is quite impressive. Each of the three types of conditional branch instruction (jump, call and return) has eight condition tests available: not zero, zero, no carry, carry, parity odd, parity even, plus, and minus.

This feature makes it possible to write very compact and efficient programs for the 8080, particularly if liberal use can be made of subroutines and nesting.

In short, although it was one of the first 8-bit microprocessors developed, the 8080 has a powerful instruction set and compares well in this respect with many of the later chips. This together with the large amount of support available on the software side still makes the 8080 a very popular device.

Intel itself has produced a number of microcomputer systems based on the 8080, ranging from quite pretentious development systems with floppy discs and in-circuit emulation facilities down to single-board systems for OEM applications. Generally these are supplied as completely wired and tested systems.

Quite apart from these is the SDK-80

System Design Kit, a small 8080 system sold as a kit and designed as a low cost evaluation and prototyping tool.

A sample of the SDK-80 kit was made available to us for evaluation by the new Australian distributors for Intel, Warburton Franki Pty Ltd. We were thus able to see how the kit goes together, and to use it to try our hand at programming an 8080-based system.

The SDK-80 provides all parts required to build a complete single-board 8080 system. The kit comes together with a set of literature which includes an 8080 user's manual, an assembly language programming manual, kit instructions and a user's guide for the SDK-80 itself, a programming reference card and other material.

When assembled the SDK-80 provides an 8080 system with 256 bytes of RAM and a 1k-byte ROM containing a monitor-debug program. A 1k-byte EPROM is also provided for user program storage. In addition, the PC board provides decoding and DIL locations for simple expansion of the system to one having 1k bytes of RAM and 4k bytes of ROM/PROM, merely by wiring in the additional memory chips.

The kit also provides an 8255 programmable peripheral interface (PPI), giving 24 bits of parallel I/O interfacing. A second 8255 is provided for on the PCB pattern, to provide a further 24 bits if desired. In addition the kit includes an 8251 USART for communication with a teleprinter, video terminal or another serial terminal. This is supported by a baud-rate generator deriving 7 standard data rates from the 8080 system clock: 75, 110, 300, 600, 1200, 2400 and 4800 baud.

The serial interfacing may be set by jumpers for either 20mA current loop, RS232-type voltage levels or TTL logic levels. All other interfaces to the SDK-80 are TTL compatible, including the 8255 parallel interface.

Two 25-pin RS232C sockets are provided on the PCB for interfacing, and matching plugs are provided. There is also provision on the PCB for address bus buffering, so that the SDK-80 may be expanded beyond the PCB via the main edge connector.

The monitor program which comes with the SDK-80, in the ROM, provides six basic commands. These are listed below, together with the command letter:

Display memory contents..... D
Move blocks of memory .....M
Substitute memory locations .....S
Insert hex code..... I
Examine registers X
Go to user program..... G

Normally the I command is used for feeding in a user program from the terminal keyboard, and the D command for checking that it has been entered cor-

rectly. The S command can be used to correct or otherwise change instructions or data in memory , and the X command to set the 8080 registers before a program is run, by then using the G command.

The M command is a rather powerful one, making it possible to move a block of instructions or data along in memory. This can save a lot of tedious re-entering, for example, if you dicover you have left out an essential instruction near the start of a program!

To my knowledge no other small microprocessor evaluation kit has a monitor program providing such a "move" command, which is generally found on more pretentious systems. So the SDK-80 is rather unique in this respect.

On the other hand most other evaluation kits have monitors which provide commands to allow a program in memory to be dumped onto paper tape or cassette, and then reloaded again next time. Strangely enough the SDK-80 monitor doesn't seem to provide commands for this purpose: while the D command could be used for dumping, the dumping format is not compatible with that used by the I command.

This seems a disappointing oversight on the part of the SDK-80 designers. I suspect most users would have been prepared to forego the luxury of the M command, if this had been necessary in order to provide for convenient dumping and loading of programs.

It took about 7 hours to assemble the SDK-80 from the kit, taking care with the soldering as the PCB has narrow conductors and closely spaced pads. I found the kit assembly instructions quite clear, although one has to be careful when it comes to fitting the various option links. The designers have provided an almost bewildering array of options, in an effort to make the kit as flexible as possible.

The finished unit requires three power supplies: +5V at 1.3A, +12V at 350mA and –10V or –12V at 200mA.

When the sample SDK-80 was completed, I connected it to the EA Video Data Terminal and turned on the power. It responded with the encouraging message "MCS-80 KIT", and gave its prompt sign (a full-stop), to show that the monitor program was awaiting a command.

It was when I tried to "talk back" that a minor problem became apparent: nothing happened!

After a little troubleshooting, the reason for the lack of communication became apparent. In their wisdom, the SDK-80 designers have set the "mark" current level for the teleprinter keyboard input at around 40mA, double the nominal 20mA figure.

With the mechanical contacts of a normal teleprinter this current level would cause no problems, but the keyboard output of the EA video terminal is an opto-coupler circuit designed to switch the nominal 20mA current. While it can cope with a moderately higher level, a current of 40mA causes its voltage drop to rise quite significantly, making it seem like the keyboard is continuously in the "space" condition.

As it happens the trouble is easily fixed. I simply changed the value of a resistor in the SDK-80 interfacing circuit (R19) from 430 ohms/1W to 1k/1W. This reduced the loop current to 20mA, and all was well.

Those with video terminals having opto-coupler interfacing like the EA design may also need to make this modification to the SDK-80.

Once this modification was done, I was able to use the SDK-80 to try out some simple 8080 programs and get them going. This proved to be quite straightforward using the various monitor commands, which are very helpful apart from the lack of dump and load facilities.

Among other things, I tried writing some programs which call the utility subroutines in the monitor ROM. There are a number of useful routines available, although the kit manual suggests that the user can only use the two terminal drivers. It also suggests a rather strange indirect way of calling these, but I found that it was possible to call both the drivers and a number of other subroutines directly.

To illustrate this, I am reproducing here a listing of a simple novelty program which echoes input from the terminal, and replies with a curt "GO AWAY, I'M BUSY!" when the user terminates a line with a carriage return. This program uses the two terminal driver routines CI and CO, calling them directly via their addresses in ROM (01D0 and 01E3).

It would probably be possible to make the program shorter by using the monitor subroutine ECHO instead of CO. However, even as it stands it should give you an idea of the power and flexibility of the 8080A instruction set, at least for this type of application. You could try writing a shorter version yourself, as an exercise.

Summing up, I found the SDK-80 kit fairly easy to assemble, and easy to get going apart from the minor complication caused by the high keyboard loop current. The resident monitor program provides a powerful set of commands, including a rarely-found block move function, although there seems to be no provision for dumping to and loading from tape.

All in all, though, SDK-80 seems a businesslike little system, and one which would make a good introduction to the 8080 microprocessor. It seems quite good value for money at the quoted price of $315 plus tax.

SDK-80 kits and other Intel 8080 systems are available from Warburton Franki Pty Ltd, who have offices in each state.

```
NOVELTY ANSWER-BACK PROGRAM FOR INTEL SDK-80 KIT
WRITTEN BY J.ROWE, ELECTRONICS AUSTRALIA APRIL 1977


1300 CD D0 01 LOOP:CALL CI     ;FETCH CHAR FROM TERMNL
1303 5F             MOV E,A    ;COPY INTO E
1304 4F             MOV C,A    ;AND INTO C
1305 CD E3 01       CALL CO    ;NOW ECHO
1308 7B             MOV A,E    ;RESTORE IN A
1309 E6 7F          ANI 7FH    ;STRIP OFF PARITY
130B EE 0D          XRI 0DH    ;WAS IT A CR?
130D C2 00 13       JNZ LOOP   ;NO -- CONTINUE
1310 0E 0A          MVI C,0AH  ;YES -- SUPPLY LF
1312 CD E3 01       CALL CO
1315 21 26 13       LXI H,2613 ;SET UP H&L AS ANSWER POINTER
1318 4E         ANSR:MOV C,M   ;LOAD ANSWER CHAR INTO C
1319 79             MOV A,C    ;COPY CHAR INTO A
131A FE 00          CPI 00H    ;CHECK IF ZERO
131C CA 00 13       JZ LOOP    ;YES -- END OF ANSWER SO RETURN
131F CD E3 01       CALL CO    ;NO -- SEND TO TERMINAL
1322 23             INX H      ;INCREMENT POINTER
1323 C3 18 13       JMP ANSR   ;AND CONTINUE
1326            ; ANSWER BUFFER BEGINS HERE
1326 47 4F 20
1329 41 57 41
132C 59 2C 20
132F 49 27 4D
1332 20 42 55
1335 53 59 21
1338 0D 0A 00                  ;ANSWER MUST END WITH A ZERO BYTE
```

Here is a short novelty program which the author wrote to run on the SDK-80 kit. Note that in 3-byte memory reference instructions, the 8080A expects the LESS significant address byte first.

The Motorola 6800

One of the most established microprocessors, the Motorola MC6800 is supported by a continuously expanding "family" of memory and specialised interfacing chips—and also by a great deal of proven software and applications experience. In this article we look at the MC6800, its main support chips, and also the recently released MEK6800D2 "Mark II" evaluation kit.

by JAMIESON ROWE

Motorola Semiconductor Products were only the second major US manufacturer to enter the microprocessor field, in 1974. The MC6800 was their initial entry, and the fact that it is still one of the market leaders—and showing no signs of giving ground—testifies to the soundness of the basic design concept. Needless to say it is now supported by a large amount of development and applications software, much of it generated by users. This inevitably increases its appeal for potential new users, compared with newer entries to the field.

It is strongly supported in another sense, too: along with the basic microprocessor chip there are a number of matching memory and interfacing chips, all designed to simplify system design. These include a 1024 x 8-bit ROM (MCM6830), a 128 x 8-bit RAM (MCM6810), a programmable 16-bit bidirectional Peripheral Interface Adaptor or "PIA" (MC6820), a programmable Asynchronous Communications Interface Adaptor or "ACIA" (MC6850), a programmable Synchronous Serial Data Adaptor or "SSDA" (MC6852), and a number of more specialised devices including some which are still in development. There are also various system housekeeping devices, including a family of hybrid clock oscillators.

As the block diagram suggests, the basic architecture of the MC6800 microprocessor chip itself is fairly straightforward. This is perhaps part of the secret of its success, although there are a number of subtle strengths which also emerge upon closer inspection.

There are only six internal working registers, one of which is a condition code or processor status register. Two of the remaining registers are 8-bit accumulators, both full primary accumulators of almost equal status. The other three registers are the program counter, an index register and a stack pointer, all three of which are 16 bits long. The MC6800 implements its stack in external RAM.

The status register has six active bits, one of which is the flag for enabling and disabling the master interrupt input. The remaining status bits are condition code flags, representing arithmetic carry and overflow, sign, zero and a bit-3 carry used for BCD arithmetic. None of the status register bits is accessible directly via external device pins.

Data and address information pass between the MC6800 and the rest of a system via two separate buses. One is an 8-bit bidirectional data bus; the other is a 16-bit address bus which gives the MC6800 the capability of directly addressing 65,536 or "65k" bytes of memory space.

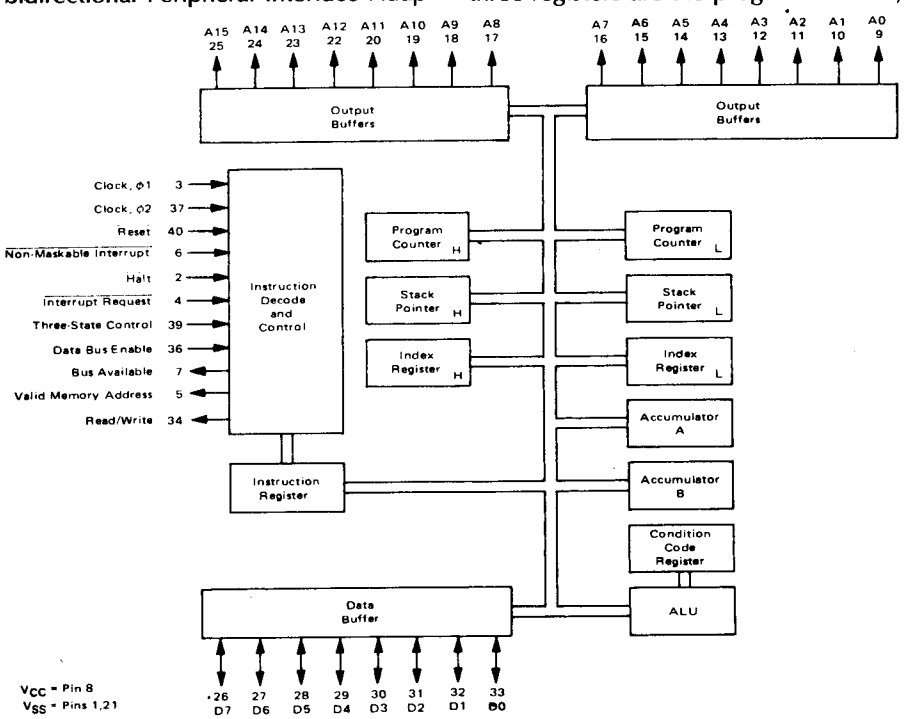
The output buffers on both the data and address lines are 3-state, and may be disabled for DMA operation.

There is no clock oscillator on the MC6800 chip itself. Instead there are two clock input pins, which must be fed with non-overlapping two-phase clock signals. Maximum clock frequency is 1MHz, and a machine cycle corresponds directly to a clock period. Fetching and execution of an instruction ranges from 2 to 12 machine cycles depending upon the instruction, or from 2 to 12us at the maximum clock rate.

Incidentally the MC6800 is an N-channel depletion mode MOS device, made using silicon gate technology. As a result it operates from a single +5V supply.

Apart from the normal reset and master interrupt inputs the MC6800 is also provided with a non-maskable interrupt input. There is also a software-interrupt instruction, for program abortion. Vectoring is provided only for the different interrupt mechanisms; within each mechanism an interrupt source must be identified by polling.

What the MC6800 does do upon being interrupted is automatically save the contents of all its registers on the stack, in external RAM. This takes place in only 10 machine cycles, far less than would be required if the interrupt routine had to perform the saving by explicit instructions (required by most other micros).



Left: Basic architecture of the MC6800 microprocessor chip. Six registers are accessible to the programmer.

The instruction set of the MC6800 comprises some 72 different instructions, ranging from one to three bytes in length. There are 29 instructions involving the accumulators and/or memory; 11 involving the pointer register and the stack pointer; 23 are jump, branch and special operation instructions; and the remainder are for status register manipulation.

The relatively large number of memory reference instructions helps to compensate for the relatively small number of working registers within the CPU chip itself. An interesting aspect of this is that many of the arithmetic and logic operations can be performed not only on the contents of the accumulators, as in other microprocessors, but on the contents of memory and interfacing chip registers as well. Thus the MC6800 lets you clear, form either the 1's or 2's complement, decrement, increment, rotate left or right, shift left or right, and test the contents of memory address and interface chip registers—as well as those of the accumulators—all with single instructions.

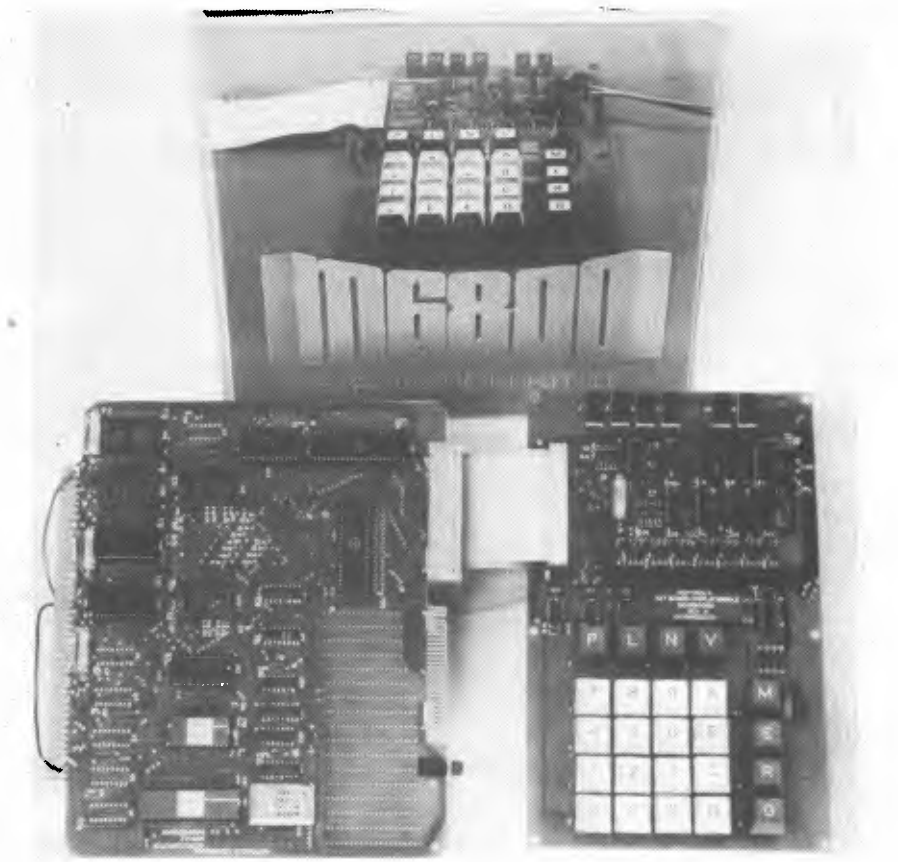
Apart from the inherent addressing used for instructions involving only accumulators and/or other internal registers, the MC6800 provides five different memory addressing modes. These are immediate addressing, direct or 1-byte absolute addressing, extended or 2-byte absolute addressing, relative addressing and indexed addressing.

Not all of these addressing modes are available for all instructions, however. In fact relative addressing is available only on branch instructions, and these instructions have no other mode available. Similarly the Jump and Jump-to-Subroutine instructions and many of the shift and rotate instructions have only indexed and extended addressing options.

With immediate addressing, the second byte of the instruction itself is interpreted as the operand data. With direct addressing the second byte is interpreted as an unsigned 8-bit absolute address, allowing addressing of the first 256 locations in memory space (00-FF hex inclusive). Extended addressing is similar to the latter except that the instruction has three bytes, and the second and third bytes are interpreted as an unsigned 16-bit absolute address, allowing addressing of any location in the 65k memory space.

In the relative addressing mode the processor interprets the second instruction byte as a signed 8-bit number, which is added to the current program counter contents to give the effective address. This allows addressing in the range from -128 to +127 bytes away from the location immediately after the second instruction byte.

In the indexed addressing mode the contents of the index register are used in generating the effective address, as with other microprocessors. However, in this case the second byte of the instruction is interpreted as an unsigned 8-bit number to be added to the index register contents, not a signed number as with the relative addressing mode. This means that the indexed address range only extends forward from the location specified by the index register, not backward; however, there is still a full 256-address range.



Here is the assembled MEK6800D2 evaluation kit. The two PCBs are shown in front of the binder containing hardware data and programming manuals.

The large number of branch instructions provided by the MC6800 allow for considerable programming economy, particularly when data is being manipulated. Conditional branching conditions include carry clear, carry set, zero, greater or equal to zero, greater than zero, higher negative or zero, negative greater than zero, lower or same, minus, not equal to zero, overflow clear, overflow set, and plus.

Incidentally the MC6800 has no separate IOT instructions; all I/O devices are accessed as locations within the 65k memory space. As the MC6800 provides some fancy memory reference instructions, this can simplify programming for complex data communication applications.

Returning to the hardware side, one of the essential devices in any system using the MC6800 is a clock oscillator. As the microprocessor requires fairly critical non-overlapping two phase clock signals, Motorola provide a family of hybrid crystal clock modules in modified 24-pin DIL compatible packages. These are the MC6870 series, some of which provide just the basic two phase signals for the CPU together with a TTL signal for memory timing, while others provide a number of other signals as well.

There are quite a few memory devices provided by Motorola to support the MC6800, including both static and dynamic RAMs, mask-programmed ROMs, and an EPROM. Most of the devices are byte-organised and provided with multiple chip-select inputs to simplify system design, while both the MCM6810 128-byte RAM and the MCM6830 ROM operate from a single 5V supply like the MC6800 itself.

Of the specialised peripheral interfacing devices in the 6800 family, the MC6820 PIA is used for parallel interfacing. It provides 16 I/O pins, grouped in two sets of 8 although all pins may be individually programmed as either inputs or outputs. Associated with each set of 8 I/O pins within the PIA are three separate 8-bit registers, making six in all.

(continued overleaf)

Two programs for the Baby 2650 Microcomputer

by PERRY BROWN

Courtesy Applied Technology Pty Ltd

GUESSING GAME

```
LOCATION  CODE          MNEMONICS
  0440    07 07       STRT LODI R3 H'07'
  0442    3F 04 B6      FZ BSTA UN PRNT
  0445    05 00         F1 LODI R1 H'00'
  0447    E5 63         F2 COMI R1 H'63'
  0449    19 7A            BCTR 'GT' F1
  044B    12               SPSU
  044C    E4 00            COMI R0 H'00'
  044E    19 02            BCTR 'GT' F4
  0450    D9 75         F3 BIRR R1 F2
  0452    E5 00         F4 COMI R1 H'00'
  0454    18 7A            BCTR '=' F3
  0456    77 10            PPSL H'10'
  0458    07 00            LODI R3 H'00'
  045A    75 10            CPSL H'10'
  045C    07 36            LODI R3 H'36'
  045E    3F 04 B6      F5 BSTA UN PRNT
  0461    07 1F            LODI R3 H'1F'
  0463    3F 04 B6         BSTA UN PRNT
  0466    3B 3B            BSTR UN INPT
  0468    07 09            LODI R3 H'09'
  046A    C2               STRZ R2
  046B    82            F6 ADDZ R2
  046C    FB 7D            BDRR R3 F6
  046E    C2               STRZ R2
  046F    3B 32            BSTR UN INPT
  0471    82               ADDZ R2
  0472    77 10            PPSL H'10'
  0474    87 01            ADDI R3 H'01'
  0476    75 10            CPSL H'10'
  0478    07 10            LODI R3 H'10'
  047A    E1               COMZ R1
  047B    19 61            BCTR 'GT' F5
  047D    07 18            LODI R3 H'18'
  047F    E1               COMZ R1
  0480    1A 5C            BCTR 'LT' F5
  0482    07 28            LODI R3 H'28'
  0484    3B 30            BSTR UN PRNT
  0486    77 10            PPSL H'10'
  0488    05 30            LODI R1 H'30'
  048A    A7 0A         F7 SUBI R3 H'0A'
  048C    E7 00            COMI R3 H'00'
  048E    1A 02            BCTR 'LT' F8
  0490    D9 78            BIRR R1 F7
  0492    87 3A         F8 ADDI R3 H'3A'
  0494    01               LODZ R1
  0495    3F 02 B4         BSTA UN COUT
  0498    77 10            PPSU H'10'
  049A    03               LODZ R3
  049B    3F 02 B4         BSTA UN COUT
  049E    07 02            LODI R3 H'02'
  04A0    1F 04 42         BCTA UN FZ
  04A3    3F 02 86   INPT BSTA UN CHIN
  04A6    E4 30            COMI R0 H'30'
  04A8    1A 79            BCTR 'LT' INPT
  04AA    E4 39            COMI R0 H'39'
  04AC    19 75            BCTR 'GT' INPT
  04AE    C3               STRZ R3
  04AF    3F 02 B4         BSTA UN COUT
  04B2    03               LODZ R3
  04B3    44 0F            ANDI R0 H'0F'
  04B5    17               RET UN
  04B6    0F 24 BF   PRNT LODA+R2 MSAG
  04B9    E4 00            COMI R0 H'00'
  04BB    14               RETC '='
  04BC    3F 02 B4         BSTA UN COUT
  04BF    1B 75            BCTR UN PRNT
  04C1    00 20 54   MSAG "NUL SP T
  04C4    52 59 53 0D      R Y S CR
  04C8    0A 52 45 41      LF R E A
  04CC    44 59 3F 00      D Y ? NUL
  04D0    0D 0A 48 49      CR LF H I
  04D4    47 48 21 00      G H ! NUL
  04D8    0D 0A 4C 4F      CR LF L O
  04DC    57 21 00 0D      W ! NUL CR
  04E0    0A 47 55 45      LF G U E
  04E4    53 53 2D 00      S S - NUL
  04E8    0D 0A 59 45      CR LF Y E
  04EC    53 21 20 41      S ! SP A
  04F0    46 54 45 52      F T E R
  04F4    20 00 0D 0A      SP NUL CR LF
  04F8    4F 4B 20 31      O K SP 1
  04FC    2D 39 39 00      - 9 9 NUL
```

When called, the program will wait until you enter any character. It will then generate a random number from one to 99 which you must guess. Start execution at 0440.

NIM

```
LOCATION  CODE          MNEMONICS
  0440    05 17       STRT LODI R1 H'17'
  0442    07 29            LODI R3 H'29'
  0444    3F 04 B3         BSTA UN PRNT
  0447    1B 0E            BCTR UN F1
  0449    07 09            LODI R3 H'09'
  044B    3F 04 B3         BSTA UN PRNT
  044E    01               LODZ R1
  044F    A2               SUBZ R2
  0450    C1               STRZ R1
  0451    02               LODZ R2
  0452    64 30            IORI R0 H'30'
  0454    3F 02 B4         BSTA UN COUT
  0457    07 1E         F1 LODI R3 H'1E'
  0459    3F 04 B3         BSTA UN PRNT
  045C    01               LODZ R1
  045D    C3               STRZ R3
  045E    04 30            LODI R0 H'30'
  0460    A7 0A         F2 SUBI R3 H'0A'
  0462    1A 02            BCTR 'LT' F3
  0464    B6 7A            BIRR R0 F2
  0466    87 3A         F3 ADDI R3 H'3A'
  0468    3F 02 B4         BSTA UN COUT
  046B    03               LODZ R3
  046C    3F 02 B4         BSTA UN COUT
  046F    E5 01            COMI R1 H'01'
  0471    19 07            BCTR 'GT' F4
  0473    07 00            LODI R3 H'00'
  0475    3B 3C            BSTR UN PRNT
  0477    1F 04 40         BCTA UN STRT
  047A    07 16         F4 LODI R3 H'16'
  047C    3B 35            BSTR UN PRNT
  047E    3F 02 86      F5 BSTA UN CHIN
  0481    E4 31            COMI R0 H'31'
  0483    1A 79            BCTR 'LT' F5
  0485    E4 33            COMI R0 H'33'
  0487    19 75            BCTR 'GT' F5
  0489    C3               STRZ R3
  048A    47 0F            ANDI R3 H'0F'
  048C    3F 02 B4         BSTA UN COUT
  048F    01               LODZ R1
  0490    A3               SUBZ R3
  0491    C1               STRZ R1
  0492    E5 01            COMI R1 H'01'
  0494    19 07            BCTR 'GT' F6
  0496    07 3B            LODI R3 H'3B'
  0498    3B 19            BSTR UN PRNT
  049A    1F 04 40      F6 BCTA UN STRT
  049D    A4 05            SUBI R0 H'05'
  049F    1A 0E            BCTR 'LT' F7
  04A1    A4 03         F8 SUBI R0 H'03'
  04A3    19 7C            BCTR 'GT' F8
  04A5    84 03            ADDI R0 H'03'
  04A7    C2            FA STRZ R2
  04A8    98 02            BCFR '=' F9
  04AA    86 01            ADDI R2 H'01'
  04AC    1F 04 49      F9 BCTA UN FH
  04AF    84 04         F8 ADDI R0 H'04'
  04B1    1B 74            BCTR UN FA
  04B3    0F 24 BA   PRNT LODA+R3 MSGE
  04B6    14               RETC '='
  04B7    3F 02 B4         BSTA UN COUT
  04BA    1B 77            BCTR UN PRNT
  04BC    0D 0A 49   MSGE "CR LF I
  04BF    20 57 49 4E      SP W I N
  04C3    00 0D 0A 49      NUL CR LF I
  04C7    27 4C 4C 20      ' L L SP
  04CB    54 41 4B 45      T A K E
  04CF    20 00 0D 0A      SP NUL CR LF
  04D3    4D 4F 56 45      M O V E
  04D7    2D 00 0D 0A      - NUL CR LF
  04DB    4E 4F 20 4C      N O SP L
  04DF    45 46 54 3D      E F T =
  04E3    00 0D 0A 0A      NUL CR LF LF
  04E7    4C 45 41 56      L E A V
  04EB    45 20 31 20      E SP 1 SP
  04EF    54 4F 20 57      T O SP W
  04F3    49 4E 00 0D      I N NUL CR
  04F7    0A 59 4F 55      LF Y O U
  04FB    20 57 49 4E      SP W I N
  04FF    00               NUL
```

The game of Nim: starting with 23, you and the machine take turns at subtracting a number from one to three. The one that leaves one after their move wins. Start execution at 0440.

One in each group is a data buffer, another a latch whose bits specify whether the device pins are used as inputs or outputs; the third is a control register used to define interfacing protocol and status. All six PIA registers are addressable in MC6800 memory space, although in a slightly confusing manner: the control registers are addressed directly, while the data and direction registers share common addresses and must be distinguished by setting a control register bit.

The other peripheral interfacing device most likely to be found in smaller 6800 systems is the MC6850 ACIA, used for asynchronous serial interfacing. The ACIA is rather like a UART, having separate sections for asynchronous transmission and reception. However unlike a UART these share a common 8-bit parallel bidirectional interface to the 6800 system data bus. In addition, there is an 8-bit control register addressable separately in 6800 memory space, which allows program control of serial data format, a choice of three communication rate clock division ratios, and the handshaking protocol. There is also a status register, sharing the same address as the control register, whose bits may be read to determine ACIA status.

Having looked briefly at the MC6800 microprocessor, its instruction set and some of its support chips, let us now turn to the new MEK6800D2 evaluation kit. This has been produced by Motorola to provide a complete low-cost 6800 system, for both evaluation and basic program development.

The MEK6800D2 is an assemble-it yourself kit, which goes together to produce two PCB assemblies. One is the microcomputer itself, on a PCB measuring 248 x 210mm overall; the other is low-cost terminal unit on a PCB measuring 254 x 159mm.

The assembled microcomputer board has the MC6800 itself, a crystal clock module (614.4kHz), a 1k byte ROM with resident "JBUG" monitor program, three 128-byte RAMs (one of which is allocated to the monitor, leaving 256 bytes for user programs), two PIA devices and an ACIA.

The PCB also has decoding and sockets for easy expansion using a further two 128-byte RAMs, and two 1k byte EPROMs (MCM68708). It also has space for data bus and address bus buffers, if the user wishes to expand further into a multi-board system.

The assembled "terminal" PCB has a 24-key keyboard and a display using six 7-segment LEDs. Together these can be used with the JBUG monitor for entering programs, examining memory and registers, single stepping through a program, setting and removing breakpoints (five

are permitted), and transferring control to the user program.

In addition, the terminal PCB contains a full audio tape interface, to allow dumping and loading of programs using a normal cassette or reel-to-reel tape recorder. All that is required apart from the kit (and power supply) are a couple of shielded leads with suitable audio connectors.

The loading and dumping operations are controlled by further JBUG routines. Transfer takes place at a rate of 300 bauds, and the format conforms to the "Kansas City Standard" with 2400/1200Hz tones. The interfacing circuitry requires no "tweaking", using a stable counter-type decoder.

The terminal PCB connects to the main microcomputer PCB by means of a 50-way flat ribbon cable and edge connector. The keyboard and LED display interface via one of the two PIAs, while the tape interface uses the ACIA. The remaining PIA on the main PCB is available for user interfacing, with the terminal connected. If the user later decides to use the kit for a dedicated application, without the terminal PCB, all three interfacing chips can be used for interfacing.

It is also possible to convert the kit over for operation with a teleprinter or other serial asynchronous terminal. The main change required is to replace the JBUG ROM with another containing the terminal-orientated monitor "MINIbug III".

The complete MEK6800D2 kit operates from a single +5V power supply, drawing about 1 amp.

The Sydney office of Motorola Semiconductor sent us one of the MEK6800D2 kits, so that we would be able to assemble it and report to readers on our findings.

The kit comes in a single box, which opens to reveal one of the large spring-clip binders. Inside are two blister packs containing most of the parts for the two PCB modules, together with a plastic bag containing the rest of the parts. There are also a number of handbooks, including a kit manual, programming reference manual and M6800 system design book.

I found it fairly easy to put the kit together, although the instructions are rather cursory and assume that the builder has a fair amount of experience. Assembly took me about 7 hours, but I wasn't trying to break any records.

The kit worked perfectly when power was applied, and I was then able to run through the introductory program load-run-debug example which Motorola have thoughtfully provided in the kit manual. This is well done, and should give a newcomer to microcomputer systems a good idea of the basic concepts of program manipulation via a monitor.

When it came to writing our own programs, the going wasn't quite as easy. The programming reference manual doesn't seem to me to be particularly well written, at least as far as the introductory material is concerned. For example the material describing the various 6800 addressing modes complicates the issue by talking quite a lot about assembly language syntax, so that a beginner could get very confused.

```
SIMPLE DISPLAY PROGRAM FOR MOTOROLA MEK6800D2 KIT
WRITTEN BY J.ROWE, ELECTRONICS AUSTRALIA  9.3.1977

0000 CE 00 24  START,LDX DISBUF     SET X AS BUFF PTR
0003 DF 22           STX XBUF       & SAVE
0005 86 20           LDA A $20      SET PIA FOR DISPLAY U1
0007 B7 80 22        STA A DISREG
000A A6 00      LOOP,LDA A 0,X      FETCH CHAR VIA X
000C B7 80 20        STA A SEGREG   & DISPLAY
000F CE 00 4D        LDX $4D        SET UP X FOR 1MS DELAY
0012 BD E0 E0        JSR DLY1       CALL JBUG DELAY S-R
0015 7C 00 23        INC XBUF+1     INCREMENT SAVED BUFF PTR
0018 DE 22           LDX XBUF       & RESTORE TO X
001A 0C              CLC            CLEAR CARRY
001B 74 80 22        LSR DIGREG     UPDATE DIGIT PTR IN PIA
001E 24 EA           BCC LOOP       CONTINUE UNTIL 6 DONE
0020 20 DE           BRA START      BACK TO BEGIN AGAIN
0022 00 00 (00 24)XBUF,             X IS STORED HERE
0024 02                             START OF MESSAGE BUFFER
0025 00
0026 40
0027 40
0028 21
0029 24
```

A simple novelty program for the MEK6800D2 kit. It displays encoded characters stored in locations 0024-0029 on the kit's 6-digit LED display.

As I was not too familiar with the 6800, it took a while to sort out chip operation from assembler operation. One thing which helped was a look through the listing for the kit's J-BUG monitor program, which Motorola have thoughfully given in the manual.

Once the addressing modes were sorted out, I was able to begin writing a few short programs and try them out. A sample program is reproduced here, as readers may find it interesting. It was written as a little exercise to see how one can use the LED display under user program control.

The kit manual doesn't help a great deal in telling you how to display data, so I had to deduce the way of doing this from the terminal circuit diagram and the J-BUG listing. It turns out that the routines in J-BUG itself are not capable of being called by user programs, as they are not self-contained subroutines. However, in any case it is fairly easy to provide a routine in one's own program, as you can see. This is largely because the PIA does most of the work.

The comments in the right-hand column of the listing should give you a fair idea how the program works. Note that the display digit multiplexing is performed by loading a 1 into one data register of the PIA, labelled "DIGREG", and then shifting the 1 along using the LSR instruction (address 001B). Similarly the actual digits are fed to the display segments by loading them into the other PIA data register, labelled "SEGREG".

The only part of the J-BUG monitor made use of by this little program is the subroutine DLY1, used to obtain the 1ms delay between displayed digits. This is used by first loading the index register with hex 4D, to specify a 1ms delay time, and then calling the subroutine at address E0E0. The two instructions involved are those with their first bytes stored in addresses 000F and 0012.

Note that the simple 6-character message displayed by this program is stored in locations 0024-0029. They are not in ASCII code, but in a code whose first 7 bits correspond to the seven display segments, in complement form. Any characters capable of being displayed on 7-segment LEDs can be shown, by working out the appropriate codes.

To summarise, the MEK6800D2 kit seems a well-designed one, and should enable those with reasonable experience at electronic kit building to build up a low-cost 6800 system. When assembled it becomes a small but businesslike development system, adequate for learning 6800 programming and working up quite respectable programs. And with plenty of provision for expansion, you can make it grow into a more elaborate system when this is needed.

In short, good value for money at the quoted price of around $240. The kit should be available from Motorola distributors, in every state.

The National SC/MP

We continue our survey of microprocessors this month with a more detailed look at National Semiconductor's 8-bit SC/MP chip. We also take a look at the low-cost SC/MP evaluation kit, which has already aroused a lot of interest, and its more powerful "big brother" the SC/MP development system.

by JAMIESON ROWE

In addition to the 16-bit PACE microprocessor which was examined in our August issue, National Semiconductor also currently produces an 8-bit chip. This is called SC/MP, which is short for "Simple Cost-effective Micro-Processor" (and pronounced "scamp"). As the name suggests, SC/MP was designed primarily for lower level applications than PACE; applications where cost effectiveness is a major consideration.

Like the PACE chip, SC/MP is an MOS/LSI device using silicon gate, P-channel technology—although it uses depletion mode devices instead of the enhancement mode devices used in PACE. A faster N-channel version of SC/MP is currently in development, and is mooted for early in the new year.

The basic architecture of SC/MP may be seen from the diagram on this page. There are a total of 10 internal registers, five of which are 8 bits in length while the other five are 16 bits long. Data input and output takes place via an 8-bit parallel bidirectional bus, while there is a separate dedicated 12-bit address bus.

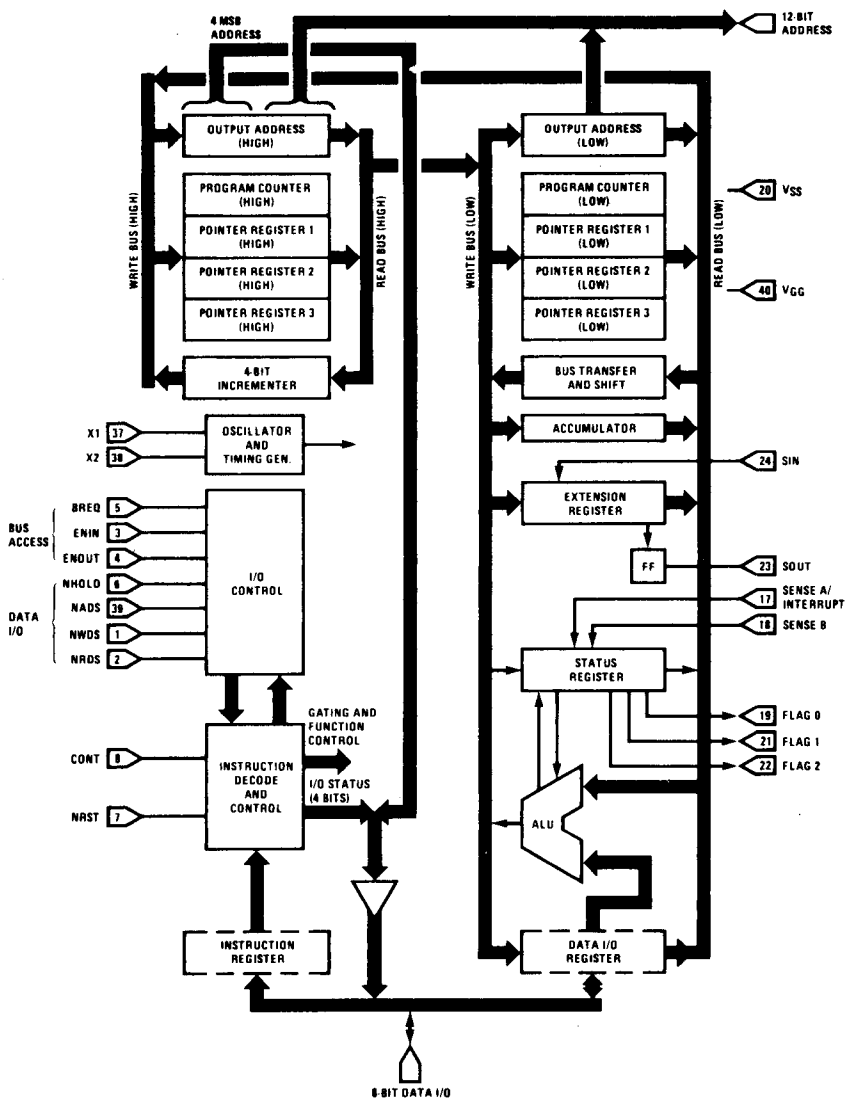
There is only one nominal 8-bit accumulator register, but this is effectively supplemented for logical and arithmetic functions by the 8-bit extension register. The other 8-bit registers are the status register, the data I/O register, and the instruction register.

Of the five 16-bit registers, one is an output address register which is transparent to the user. The remaining four are pointer registers, one of which is dedicated as the program counter. The other three are available to the programmer as addressing pointers.

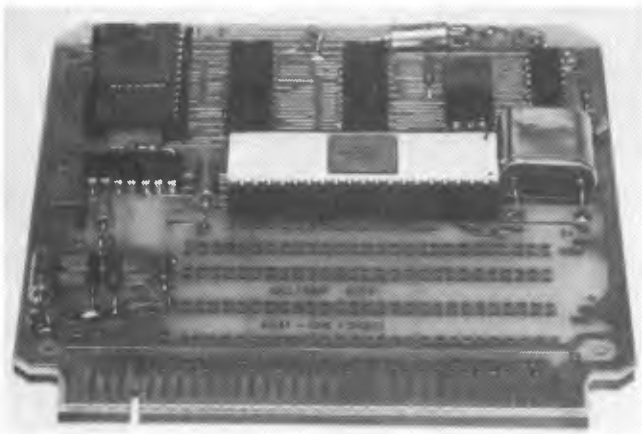
As the program counter and pointer registers are 16 bits long, this means that SC/MP has the ability to directly address 65,536 bytes of memory, or "65K". However the chip's address bus handles only the 12 least significant bits, the remaining four bits being multiplexed out of the chip on four of the data bus lines. In addition there is no carry-over to the four most significant bits when the program counter (PC) is incremented, so that memory space is effectively divided into sixteen pages of 4096 bytes each.

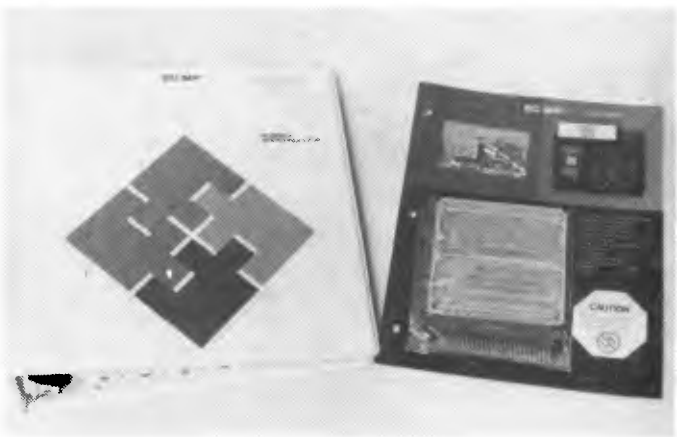
Of the 8 bits in the status register, three are used for user flag signals which are made available at device pins. Another two bits are sense bits, again brought out to device pins. One of the sense inputs also serves as an interrupt input if another of the status register bits is set to a 1. The remaining two bits are an overflow bit and a carry/link bit, used for arithmetic and shift-rotate functions.

The SC/MP chip provides five control signals to facilitate data input-output and bus accessing. These are address strobe (NADS), read strobe (NRDS), write strobe (NWDS), bus request (BREQ) and enable outputs (ENOUT). There are also five control inputs, namely reset or initialise (NRST), continue (CONT), bus enable (ENIN), bus busy (BREQ) and extend I/O cycle (NHOLD). Note that the BREQ pin is used for both input and output of control signals.



At left is the basic architecture of the SC/MP chip, taken from the maker's data.





Above is the low-cost SC/MP evaluation kit as it comes, while at left is a view of the assembled PC board. An edge connector is supplied.

Many of the control signals provided by the SC/MP chip are not required for simple systems, but are used in more elaborate systems for such purposes as direct memory access (DMA) and similar operations.

SC/MP has its own internal clock oscillator, which can use either a capacitor or a quartz crystal for timing as required. Alternatively the oscillator can be disabled and the chip fed from an external clock source, via the two timing terminals.

In short, then, the SC/MP chip is quite a flexible 8-bit microprocessor. While designed primarily to make it capable of forming the heart of low cost minimal-device controllers and dedicated systems, it is also provided with many of the facilities required for more elaborate systems.

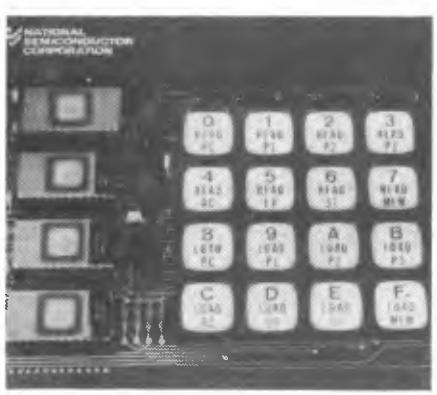
On the software side, SC/MP has a repertoire of 46 basic instructions of which 22 are 2 bytes in length, and the rest are a single byte. Fourteen of the 2-byte group are memory reference instructions, and comprise load, store, AND, OR, exclusive-OR, decimal add, add, complement and add, increment and load, decrement and load, jump, jump if positive, jump if zero, and jump if not zero.

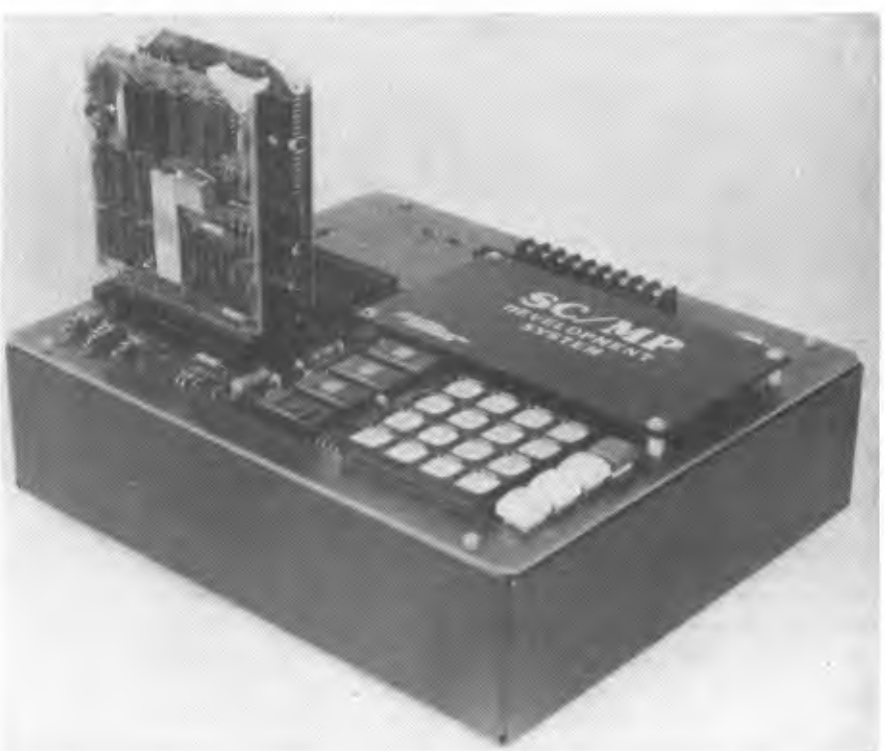
Each of these fourteen instructions may use one of three addressing modes. These are PC-relative addressing with a signed 8-bit displacement (giving a range of from -128 to +127, decimal); indexed addressing using any of the three pointer registers, and again with a signed 8-bit displacement; and auto-indexed addressing where the pointer register is either post-incremented or pre-decremented depending upon the sign of the 8-bit displacement.

In addition to these normal memory reference instructions there are no less than seven immediate-addressing instructions, wherein the data or mask operand is contained in the second byte of the instruction itself. The seven instructions are load immediate, AND immediate, OR immediate, exclusive-OR immediate, decimal add immediate, add immediate, and complement and add immediate.

Note that both the normal complement-and-add instruction and the complement-and-add immediate instruction may be used for 2's complement subtraction, by setting the carry/link bit beforehand.

The remaining double-byte instruction is a delay instruction. This may be used to provide programmable delays of from 13 to 131,593 machine microcycles, or from 26 microseconds to 263.186 milliseconds with a 1MHz clock. The delay is a function of both the displacement in the second byte of the





At right is a view of the larger SC/MP development system, with a close-up of its keyboard and EPROMs shown above.

instruction itself, and the current AC content.

Of the single-byte instructions, eight provide arithmetic and logic operations involving the accumulator and extension registers. Three more provide pointer register move operations, to cover pointer loading and PC-pointer exchanges. Five more provide shift, rotate and serial I/O operations, while the remaining eight provide miscellaneous instructions: halt, clear and set carry, interrupt enable and disable, copy status to AC and vice-versa, and NOP.

Overall, our impression of SC/MP's instruction repertoire is that it is quite a powerful one, particularly in the arithmetic and logic function area and that involving immediate instructions. These plus its "special features" such as the delay instruction and the inbuilt serial I/O would make it very well suited to dedicated computer and controller applications.

It would be less suitable for general-purpose computer applications, due to a number of limitations. One is the lack of true absolute addressing. Another is the relatively clumsy way it handles subroutine and interrupt routine servicing; this involves setting up one of the three pointer registers, and then performing a PC-pointer exchange to both enter and leave the routine. Apart from tying up a pointer register each time, this also means that the machine leaves a subroutine with its exit address in the pointer, so that subroutines which are to be called over and over must effectively have their exit immediately preceding their entrance!

Another minor shortcoming is that no shift left or rotate left instructions are provided.

Of course SC/MP was presumably not designed for general-purpose computing, so these criticisms must be judged accordingly. It is also true that given almost any instruction set one can write almost any program, once one gets fully familiar with its strengths and weaknesses. With most of us the main limitation on our programming is likely to be our own skill, whatever the chip we elect to use!

Having looked at the SC/MP chip itself, let us now look at the two small SC/MP systems which are currently available from National Semiconductor. In Australia these are available from NS Electronics Pty Ltd, or in one-off quantities through their distributors.

The system which is more widely known at present is the low-cost "SC/MP Kit", which is a minimal-system do-it-yourself evaluation kit. It comes as a large ring binder containing a complete set of parts in a blister pack, together with full assembly and programming information.

The assembled kit forms a very basic system, but one which is quite sufficient to allow development of small programs. It provides 256 bytes of RAM, together with 512 bytes of ROM containing "Kitbug"—a very basic monitor-debug program. The kit provides a 1MHz crystal for the SC/MP clock, together with the interfacing circuitry required to communicate with an ASCII teleprinter or similar 110-baud asynchronous serial terminal using a 20mA current loop.

The kit requires an external power supply providing +5V at approximately 350mA, and -12V at 200mA.

There are only three commands recognised by Kitbug: Type (T), Modify (M) and Go (G). The T command causes printout of successive memory locations and their contents, in hexadecimal, until a keyboard input is detected. The M command is similar, except that it types out a single location and provides for modification of the contents followed by either a return to Kitbug, or progression to the next location. This is the command used to load programs into the kit, and also the means for setting up the starting address and initialising the SC/MP registers prior to running. Finally the G command transfers control to the user's program, to run it.

A fourth command can be simulated, namely a user's program halt which transfers control back to Kitbug. This may be done only where the user's program does not disturb the contents of pointer register P3, making use of the fact that Kitbug stores its own return address in that register when it transfers control to the user's program. Hence by inserting an "exchange PC with P3" instruction in the user's program, control may be transferred back to Kitbug when required. This can be used as a simple breakpoint system when debugging.

Price of the low-cost evaluation kit is currently quoted as $79.95 plus tax, making it the lowest cost microprocessor evaluation kit available and the cheapest possible way of acquiring practical experience with microprocessors.

The more elaborate of the two SC/MP systems currently available in Australia is the Low Cost Development System or "LCDS", which as the name suggests is

EXCELLENT TEXT ON MICROPROCESSORS

AN INTRODUCTION TO MICRO-COMPUTERS, by Adam Osborne. Published by Adam Osborne and Associates, Inc, 2950 Seventh St, Berkeley, California. Soft covers, 133 x 206mm, about 380pp. Price in US $7.50.

In the last year, this book has apparently broken just about all previous US sales records for technical books. Its first printing sold out in a matter of weeks, and we had to wait a couple of months to get a copy of the second printing for review.

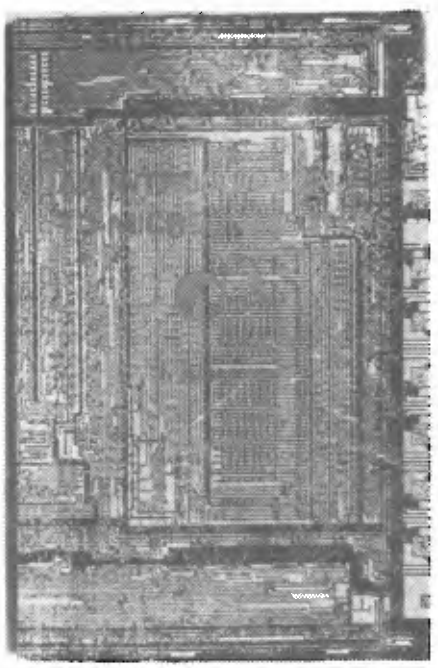
Already it has become the standard introductory text on microprocessors, and is used by many factory-run seminars and courses as an accompanying text.

Actually we understand that due to the enormous response, the original book has just been expanded into two separate volumes. By the time this review appears, the two volume version may well be the only one available.

The original book is divided into two sections: an introductory section which leads the reader through all the basic concepts of microprocessors in general, and then a survey of specific chips and their features.

Apparently the new 2-volume version adopts the same approach, but by giving each of the two sections a volume of its own, they have both been treated in greater depth.

The original book is very well written, and has been widely acclaimed. The new 2-volume version should thus be even



better, and an excellent book for anyone looking for a good up-to-date text on microprocessors and microcomputers.

We understand that Dick Smith Electronics have supplies of both volumes of the new version on order, and expect supplies by mid-October. Applied Technology have indicated that they are also obtaining stocks shortly.

Local prices for the two new volumes look like being about $7.50 each, which should still be good value for money.

```
NOVELTY ANSWER-BACK PROGRAM FOR SC/MP KIT. D.ROWE, E-A 31.7.76

200 C4 01     LDI 01        /SET UP P3 FOR CALLING GETC
202 37        XPAH P3
203 C4 65     LDI 65
205 33        XPAL P3
206 3F        XPPC P3       /GO LOOK FOR INPUT CHAR,ECHO
207 E4 0D     XRI 0D        /CR? (AC WILL HAVE ZERO IF YES)
209 9C FB     JNZ (PC)      /NO; KEEP GOING
20B C4 C4     LDI C4        /YES;SET UP P3 FOR CALLING PUTC
20D 33        XPAL P3
20E C4 0A     LDI 0A        /GO PRINT LF
210 3F        XPPC P3
211 C4 02     LDI 02        /SET UP P1 AS ANSWER POINTER
213 35        XPAH P1
214 C4 25     LDI 25
216 31        XPAL P1
217 C1 00     LD (P1)+0     /FETCH ANSWER CHAR
219 3F        XPPC P3       /PRINT
21A C5 01     LD @(P1)+1    /FETCH CHAR AGAIN,INCREMENT PTR
21C E4 0D     XRI 0D        /CR?
21E 9C F7     JNZ (PC)-9    /NO; KEEP GOING
220 C4 0A     LDI 0A        /YES; PRINT LF
222 3F        XPPC P3
223 90 DB     JMP (PC)-37   /RETURN TO LOOK FOR NEW INPUT

225 47 4F 20                /START OF ANSWER BUFFER
228 41 57 41
22B 59 2C 20
22E 49 27 4D
231 20 42 55
234 53 59 21
237 0D                      /ANSWER MUST END WITH A CR

TO RUN IN SC/MP KIT, PUT SA OF 200 IN PC STORAGE LOCATIONS 2F7 AND
2F8, THEN GIVE "G" COMMAND TO KITBUG
```

Written for the low-cost SC/MP evaluation kit, this simple answer-back program utilises the TTY servicing routines in the Kitbug ROM.

intended not so much as an evaluation kit, but as a small system for the development of programs and application systems.

The basic LCDS consists of a base supporting a large PC mother board, on which are mounted four 72-way edge connectors together with a small keyboard array, a display panel with six 7-segment LED readouts, and various other ICs including four 512-byte PROMs with a resident monitor-debug program. With the assembly comes a plug-in PC card which provides the SC/MP CPU, its clock, a 256-byte RAM and a full complement of buffers for all data, address and control lines.

Hooked up to a +5V/-12V power supply, the basic LCDS forms a limited but self-contained system using the on-board keyboard and LED display for communication. However the resident monitor-debug firmware also has provision for asynchronous serial interfacing, so that it is only necessary to connect up a teleprinter or similar terminal in order to begin more serious development work.

In teleprinter mode, the LCDS resident debug program has a powerful set of functions. It can type out the contents of all SC/MP registers, alter any of these contents at will, type out memory locations singly and modify these at will, type out sections of memory, set or remove a breakpoint, punch a selected range of memory into paper tape, load a program into memory (either the original locations, or into another area as required), and initiate execution.

The basic LCDS thus forms a more powerful "big brother" to the small evaluation kit, and as such is much more suitable for serious development of SC/MP programs.

Of course the 256-byte RAM provided on the basic CPU card is a limitation when it comes to developing larger programs. Accordingly National make available two further types of plug-in card, which may be purchased separately and plugged in as required to expand the system. One card provides an additional 2k bytes of RAM, while the other provides sockets for up to 4k of ROMs or PROMs. Both cards may be programmed using wire links to respond to a specific range of addresses in the total SC/MP memory space, so that a number of cards may be used to expand the system as far as required.

Price of the basic LCDS system complete with the CPU card is currently quoted at $405 plus tax. The optional 2k byte RAM cards are quoted at $130 each plus tax, while the 4k ROM/PROM card (less ROMs or PROMs) is $100 plus tax.

NS Electronics kindly made available to us one of the SC/MP kits for evaluation, and the author was able to put it together and gain "hands-on" experience with a minimum SC/MP system.

The assembly instructions which come with the kit are quite explicit, and I found no difficulty putting it together. Just to be safe I observed the usual precautions when soldering in MOS devices, but even this is really only necessary with the RAM devices because both the SC/MP chip itself and the ROM are provided with sockets.

In operation the Kitbug program provides pretty basic debug facilities, but once you get used to it you can develop small programs fairly easily. Out of interest I developed a novelty "answer-back" program, to duplicate the one I did for the 2650 system described last month. Again this was interesting, as the program makes use of the teleprinter servicing routines in Kitbug.

The program went together fairly easily, once I got used to the slightly tricky aspects of SC/MP subroutine servicing. It is reproduced here in the article so that readers can try it out if they feel so inclined.

NS Electronics also very kindly loaned us one of the LCDS development systems, partly to allow us to compare it with the smaller system, and partly to assist in our development of a project we have been putting together based on a SC/MP kit.

The LCDS system was complete with one of the 2k-byte RAM cards, and when I "fired it up" the increased flexibility compared with the small kit quickly became apparent. Not surprisingly, it is much easier to develop programs when you can dump and load them conveniently using paper tape, and when you can quickly insert and remove breakpoints in the program to check out sequences and isolate bugs.

In fact I found the LCDS a very convenient development system, and one which makes development of SC/MP programs a very efficient and smooth business. At the price asked, it should be a very attractive investment for anyone intending to do serious development of SC/MP programs and systems.

Both the SC/MP evaluation kit and the LCDS system components should be available in all states via NS Electronics distributors. If you require further information, this should be available from NS Electronics at either Cnr. Stud Road and Mountain Highway, Bayswater, Victoria 3153, or 2-4 William Street, Brookvale, NSW 2100.

National Semiconductor's

PACER is a complete microcomputer system designed around the National Semiconductor PACE 16-bit microprocessor chip. Housed in an attractive desk top mounting case, PACER features inbuilt input/output facilities, and is easily programmed by anyone with a little experience in computers.

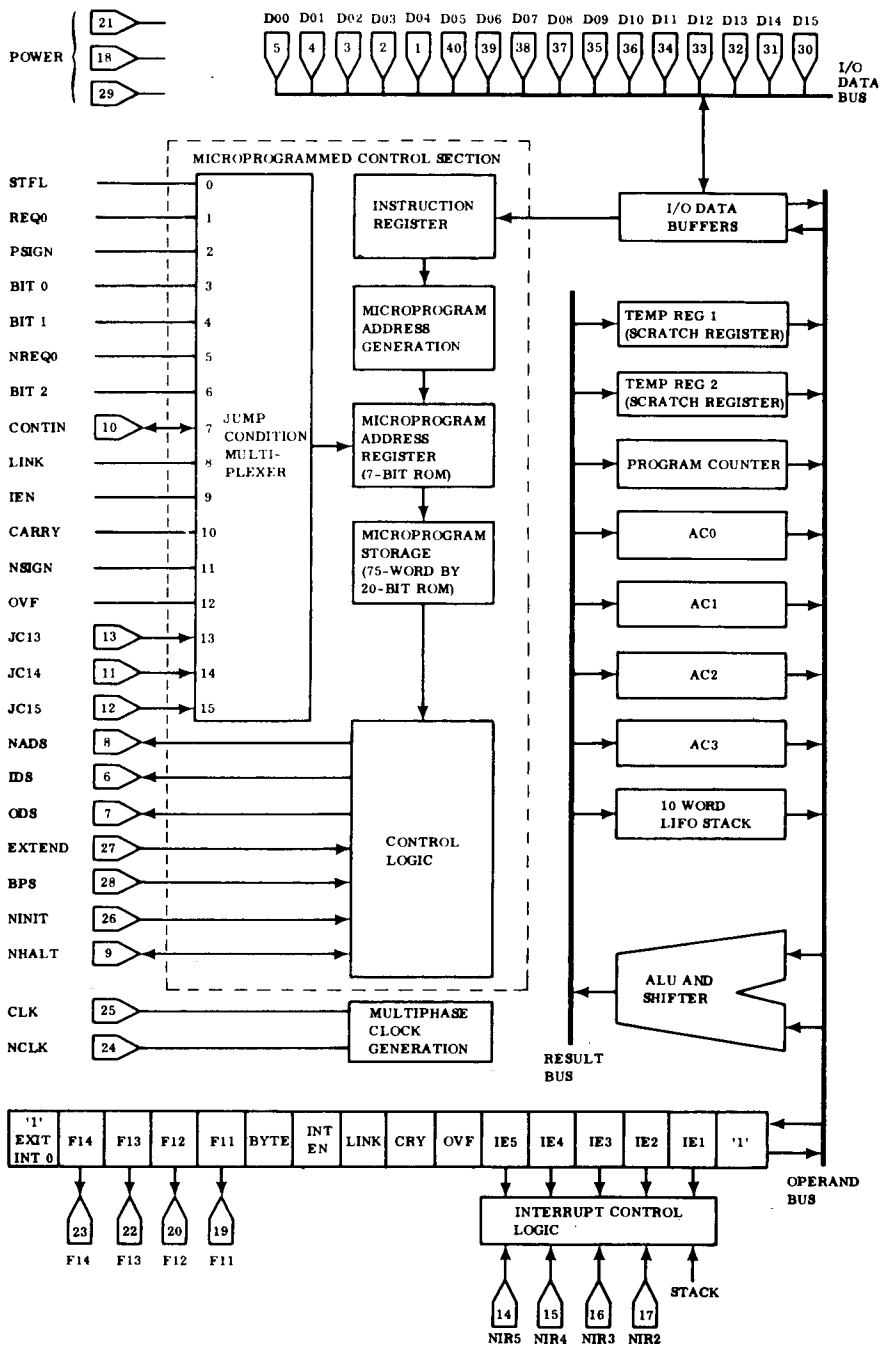


Fig. 1. The complexity of the PACE chip can be seen in this functional block diagram.

In the fast developing world of microprocessors the manufacturers of the actual microprocessor semiconductors soon realised that they had to be marketed in a different way to that for simple logic functions such as TTL or CMOS. One approach adopted (among many) was for the microprocessor manufacturer to develop a small general purpose printed circuit board containing the actual microprocessor and the necessary support hardware/software. The idea was that the prospective user could easily develop a simple system using the microprocessor card, by just adding the required power supplies and input/output devices.

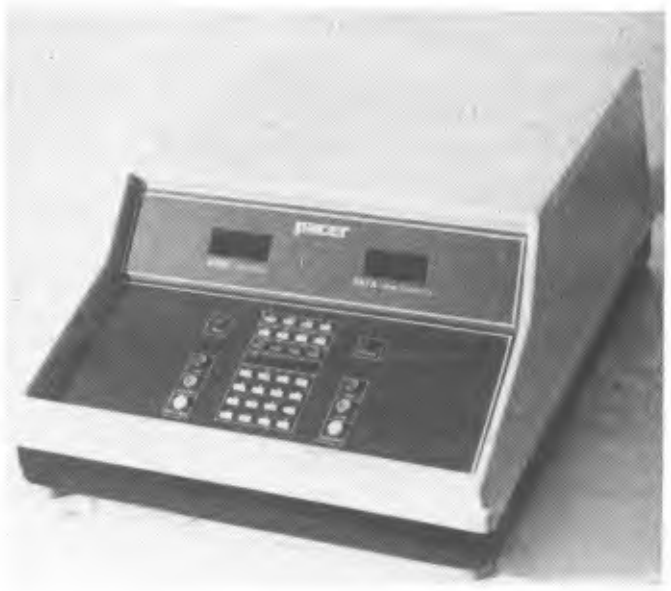
These boards, with the addition of power supplies, memory facilities and a control panel can be used as minicomputers in their own right, with the control panel LEDs being used as output devices, and the control panel switches as input devices.

As a sole source of input/output this "bit by bit"approach soon becomes quite tedious and additional input/output devices are required. This generally results in a considerable escalation of costs, particularly if a teleprinter or similar device is added. It is at this point that the approach adopted in PACER becomes a more cost effective solution.

PACER makes clever use of a 32 key keyboard with 6 additional functional keys. Under control of internal software contained in ROM, the user can easily talk to the machine in hexadecimal format, thereby considerably easing the data input operation. Yet another ingenious feature is the use of alphanumeric displays logically grouped and internally controlled so that they can display address, data, or even function in an easily read alphanumeric format.

The overall result is that it is very easy

PACER



by OWEN J. HILL, BE
*Managing Director, Applied Technology Pty Ltd

to "talk" to PACER, because the operator/machine interface is greatly improved without the need for other expensive input/output devices. Consequently, as the manufacturers claim, PACER is one of the easiest microprocessor systems to understand and use available on the market today.

Before proceeding with a more detailed examination of the PACER system, we should first look at the "heart" of the unit: the PACE microprocessor chip itself.

This chip was developed and is manufactured by National Semiconductor in the USA. It contains on one semiconductor chip all the necessary buffers, registers, control logic, memory facilities, arithmetic unit and data buses to form a 16 bit processor unit. The PACE MOS/LSI chip is produced using silicon gate P-channel enhancement mode standard process technology, which the manufacturers claim offers many advantages over other technologies. Some idea of the complexity of this device may be gained from Fig. 1.

Some outstanding features of the PACE chip are:

- 16 bit instruction word offering addressing flexibility and speed.
- 8 or 16 bit data word interfaces to increase the applications flexibility.
- 45 instruction types for efficient programming.
- Common memory and peripheral addressing.
- Four general purpose accumulators to reduce memory data transfers.
- 10 word push down/pull up stack for interrupt processing and word storage.
- Six vectored priority-interrupt levels to speed the interrupt service and simplify hardware.
- Programmer–accessible status register.

PACER is a complete system using PACE, packaged ready to use. Provision has been allowed for memory expansion, as well as a number of peripheral interface options, such as teleprinters and cassette drives. It is available in kit form, partially assembled or fully assembled and tested.

We have only seen the completely assembled and tested version, and therefore cannot comment on the kit versions. The sample was supplied with an instruction manual and a set of circuit and assembly drawings.

The accompanying pictures give a good idea of the internal construction employed in PACER. A "mother" board is used to make interconnections between a number of plug-in printed circuit boards, and also to supply unregulated power to the boards, which have their own regulators. The keyboard and display assemblies connect to the mother board via a length of flat ribbon cable.

A large transformer is mounted on the mother board, along with its associated rectifiers and filter capacitors. A cooling fan is fitted to the rear of the unit. This appears to be large enough to cope with the full complement of boards which it is possible to use with the mother board, when the PACER is expanded from its basic form as supplied.

The basic unit is equipped with three plug-in printed circuit boards. These are–

1. CPU board containing PACE microprocessor with support chips and input/output buffers.
2. Control board containing the Control

This is the PACER control panel. Hexadecimal numbers are entered via the lower 16 keys, while the upper 16 are used for control. The alphanumeric displays are at the top.

program in 1k x 16 ROM with 256 x 16 control RAM.
3. A RAM/PROM board which contains 256 x 16 RAM (expandable to 1k x 16) and space for 1k x 16 PROM.

As we have already mentioned, the major benefit of PACER lies in its built in input/output facilities. This means that PACER is an ideal instructional tool and should have great appeal to educational institutions, because it can readily demonstrate computer basics, programming and mathematical manipulations.

The PACER system can readily be expanded using optional modules to interface with teleprinters, cassettes and other keyboards. The memory can also be expanded with additional RAM, or even a disc file.

In use, PACER is very powerful, yet at the same time quite limited. Essentially it operates in two modes, "debug" and "run". In the debug mode, the control program stored in ROM accepts data from the keyboard as an input, and uses the display as an output. This program is used to interface with the memory, so that user programs can be easily entered, modified and executed.

The contents of any computer register or memory location can be readily recalled and examined or modified as necessary. It is only a single keystroke operation to examine/modify the current location and repeat this process for the next sequential or even previous sequential location.

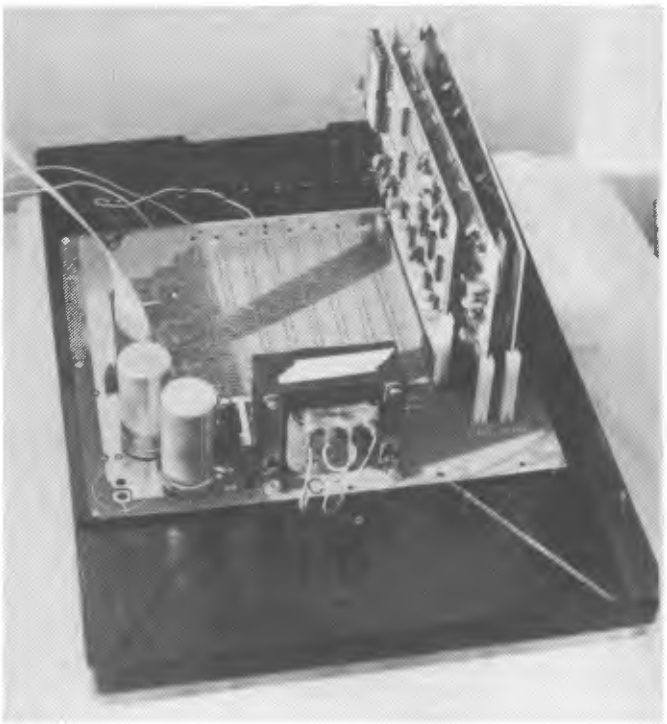
A word scan facility can be used to scan through the computer registers or memory until a location is found having a particular content. The keyboard and display can be used as a hexadecimal calculator, with entries in either decimal or hexadecimal, or from the current address. This makes the calculation of jump instruction displacements very simple.

Up to 10 breakpoints may be inserted in a user program, to return control to the debug program. Alternatively, the user program can be interrupted at any time, and the current address displayed. These facilities greatly simplify program corrections.

In the run mode, a user program is executed, starting at a particular location. A green LED indicates that a program is running. If desired, the program can be advanced one step at a time, with control returned to the debug program after each step.

An intitial attempt at writing and using a small program, however, soon pointed up a few omissions from the PACER literature, as well as a major limitation of the machine itself.

While a reasonable explanation of the way in which the debug program operates is given, the only guide to programming is a very sparse list of the PACE instruction set. There is only one very small, and very simple user program supplied.



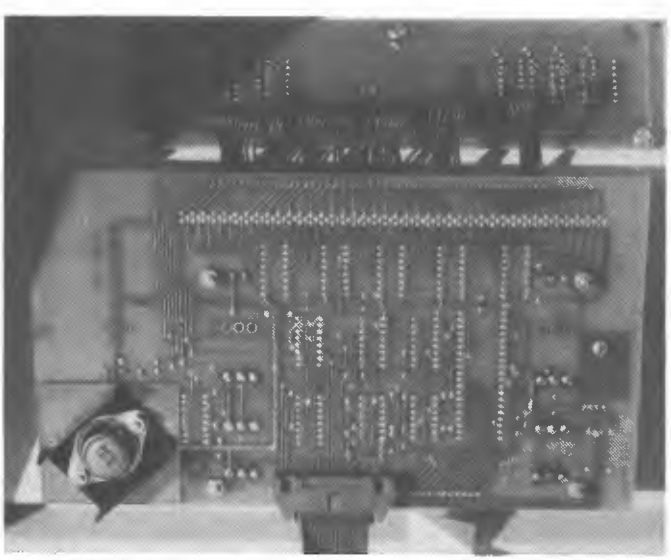
The internal construction of PACER can be clearly seen in this photograph. The mother board at the bottom is used to interconnect the remaining boards, and to supply them with unregulated power. Note the expansion capability.

The remainder of the instruction manual is concerned with interfacing to peripheral devices. This is of little use with the basic PACER system, which has no peripheral interfacing. The section on hexadecimal notation would be quite helpful for those unfamiliar with this notation, but more material on basic programming would be very worthwhile.

The major limitation of PACER as supplied, however, is that a user program cannot easily gain access to the keyboard and display for use as makeshift peripherals. A user trap has been incorporated in the debug program to prevent this, and the instruction manual does not tell how this can be circumvented.

A second limitation is that no listing is provided of the debug program itself, so that it is virtually impossible to use any of the routines existing in it. This means that even though we did manage to gain access to the keyboard and display, we had to write our own servicing routines.

Access to the keyboard and display is, in fact, obtained by altering a connection on the CPU board. Pin 8 of IC "B5" (a 7402 TTL NOR gate) must be isolated from the track which connects to it, and connected directly to the adjacent earth



The upper board holds the alphanumeric displays, while the lower one holds the keyboard and associated components. The ribbon cable connects to the mother board.

track (connected to pin 7). Both these tracks are on the top of the board, adjacent to pin 8. This alteration in no way modifies the normal operation of the machine.

Once this modification is done, access to the keyboard may be obtained via memory location DFE3. Data from the keyboard is placed in this location by the hardware and debug software. Similarly, data stored in location DFE5 is accepted by the hardware and debug software and transferred to the display. Both these locations must be addressed indirectly. We understand that National Semiconductor will in future be supplying listings of the routines and codes necessary to use the display and keyboard, as well as a listing of the debug program.

As a guide to beginners, David Edwards of Electronics Australia staff has written a small program which uses the keyboard as an input device, and the display as output. This program, which is by no means optimised as far as speed, simplicity and use of memory space is concerned, does give an indication of how PACER can be programmed.

Fig. 2 is a listing of the program, as it would be printed out by a teleprinter under computer control. Locations 0000 to 0020 contain the program, while locations 0060 to 007F are used for storage of constants, and also a small subroutine.

Note that the LOCATION and CODE columns are all that need to be fed into PACER, the MNEMONIC and COMMENTS columns are intended only to aid the programmer in understanding the operation and use of the program.

The program starts at location 0000, and will operate continuously, provided the INIT and RESTART keys are not depressed. These will halt the program, which must then be restarted.

Readers will notice that we have not explained just what the program does! This can be deduced by studying the program. Alternatively, PACER is on display at Applied Technology Pty Ltd at 109-11 Hunter St, Hornsby, NSW, and interested readers may see the program in operation there.

Although this program may seem to have somewhat little value, it is easy to see how variations can be developed so that you can play various games against the computer. In fact, with some more memory added to the basic system it is quite easy to program PACER to become a fully operational four function calculator. This readily demonstrates the long term purpose of microprocessors, i.e., using just software it is possible to program a basic general purpose module to carry out the same function as dedicated, hardwired systems in use today.

PACER DEMONSTRATION PROGRAM. BY DAVID EDWARDS. 21/6/76

LOCATION	CODE	MNEMONIC	COMMENTS
0000	A07F	LD@	LOAD A0 WITH (007F)
0001	4B01	BOC	TEST CHARACTER
0002	1800	JMP	JUMP BACK TO START
0003	51EC	LI	LOAD A1 WITH FFEC (-20 DEC)
0004	526A	LI	LOAD A2 WITH FIRST LOC OF MESSAGE
0005	C200	LD	LOAD A0 WITH (A2)
0006	5F00	RCPY	COPY A0 INTO A3
0007	2C10	SHR	SHIFT A0 RIGHT BY 8, LOAD WITH 0'S
0008	6300	PUSH	STORES (A3) IN STACK
0009	CC69	LD	LOAD A3 WITH (0069)
000A	68C0	RAD	ADD A3 AND A0, RESULT TO A0
000B	B07E	ST@	STORE A0 TO DFE5
000C	1464	JSR	JUMP TO DELAY
000D	6400	PULL	RETRIEVE (A3) FROM STACK
000E	B07E	ST@	STORE A0 TO DFE5
000F	1464	JSR	JUMP TO DELAY
0010	C063	LD	LOAD A0 WITH (0063)
0011	F062	SKNE	COMPARE A0 WITH (0062), SKIP IF NOT =
0012	1814	JMP	JUMP TO 0014
0013	D069	ST	STORE A0 TO 0069
0014	7A01	AISZ	INC A2 BY 1
0015	7901	AISZ	INC A1 BY 1
0016	1805	JMP	JUMP TO 0005
0017	C062	LD	LOAD A0 WITH (0062)
0018	D069	ST	STORE A0 TO 0069
0019	C060	LD	LOAD A0 WITH (0060)
001A	7801	AISZ	ADD 1 TO A0, SKIP IF ZERO
001B	1821	JMP	JUMP TO 0021
001C	C061	LD	LOAD A0 WITH (0061)
001D	D060	ST	STORE A0 TO 0060
001E	5000	LI	CLEAR A0
001F	B07E	ST@	STORE A0 TO DFE5
0020	1800	JMP	JUMP BACK TO START
0021	D060	ST	STORE A0 TO (0060)
0022	1803	JMP	JUMP TO 0003
0060	FF80		DISPLAY TIME COUNTER
0061	FF80		DISPLAY TIME
0062	8000		TRIGGER NO 2
0063	0000		TRIGGER NO 1
0064	C068	LD	START DELAY: LOAD A0 WITH (0068)
0065	7801	AISZ	ADD 1 TO A0, SKIP IF ZERO
0066	1865	JMP	JUMP TO 0065
0067	8000	RTS	END DELAY: RETURN FROM SUBROUTINE
0068	FFF0		DELAY LENGTH
0069	8000		TRIGGER
006A	0000		MESSAGE
006B	0000		"
006C	003E		"
006D	4149		"
006E	492E		"
006F	3E41		"
0070	4141		"
0071	3E00		"
0072	0000		"
0073	0000		"
0074	3F48		"
0075	4848		"
0076	3FFF		"
0077	0204		"
0078	027F		"
0079	3F48		"
007A	4848		"
007B	3F70		"
007C	080F		"
007D	0870		"
007E	DFE5		DISPLAY ADDRESS
007F	DFE3		KEYBOARD ADDRESS

GENERAL COMMENTS: START AT LOCATION 0000, PROGRAM RUNS CONTINUOUSLY. DO NOT TOUCH RESTART AND INIT KEYS.

This program uses the keyboard as an input device, and the display as an output device. Can you deduce what it does?

The Signetics 2650

We continue our survey of microprocessor chips and systems with this article, which takes a more detailed look at the Signetics 2650 device and its currently available evaluation kits. Although a relatively recent entry into the market, the 2650 has a particularly powerful instruction set and very flexible interfacing requirements. It seems likely to become the preferred 8-bit device for general purpose microcomputers.

by JAMIESON ROWE

The Signetics 2650 is an 8-bit microprocessor which is made using well proven N-channel MOS technology. It runs from a single +5V supply, which tends to simplify power supply requirements. All inputs and outputs are TTL compatible, and the chip requires only a single-phase clock signal input. As the chip operates in static mode, there is no minimum clock frequency.

With the original 2650 chip, the maximum clock frequency was 1.25MHz, giving instruction cycle times of from 4.8 to 9.6 microseconds. However, the currently available 2650-1 chip is rated to operate up to 2MHz, reducing the instruction cycle times to the range 3.6 – 7.2us.

The broad architecture of the 2650 chip is shown in the block diagram below. It uses an 8-bit bidirectional data bus, and a separate 15-bit address bus. This gives a direct addressing range of 32,768 bytes (32k), arranged in four pages of 8,192 bytes.

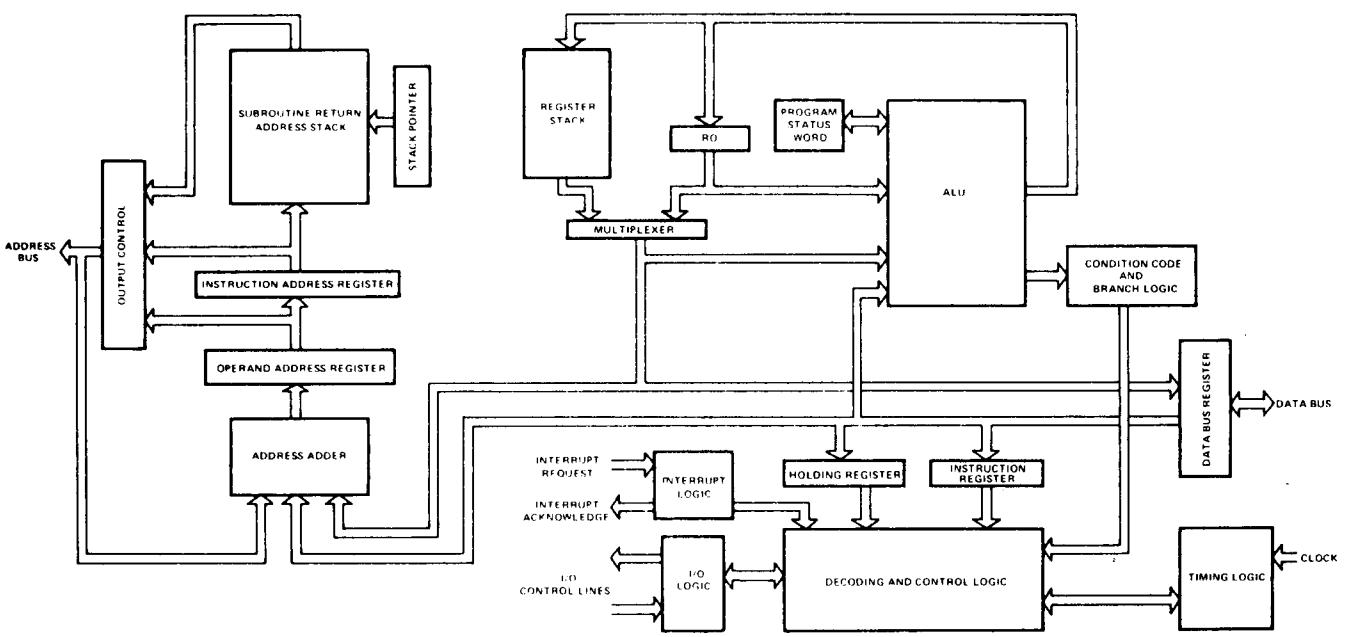
There are seven 8-bit addressable general purpose registers, one of which is the accumulator R0. The remaining six make up the register stack, and are arranged in two groups of 3 selectable by one of the bits in the Program Status Word register (PSW).

Apart from the register stack there is a subroutine return address stack, consisting of eight 15-bit registers. This allows storage of up to eight return addresses, and hence provides for up to eight levels of subroutine nesting. (A nested subroutine is a subroutine itself called by a subroutine.)

The arithmetic and logic unit (ALU) performs arithmetic, logic and shifting functions. It operates on 8-bits in parallel, and uses carry-look-ahead logic. A second adder is used to increment the instruction address register (program counter), and also to calculate operand addresses for the indexed and relative addressing modes. This separate address adder, together with the separate instruction address and operand address registers allows complex addressing modes to be implemented with no increase in instruction execution time.

The PSW is a special purpose register which contains status and control bits. The PSW bits may be tested, loaded, stored, preset or cleared using instructions which affect the PSW. Three of the bits act as the pointer for the return address stack; two others act as a "condition code" register, which is affected by the results of compare, test and arithmetic instructions and may be used by conditional branch instruction; other bits store overflow, carry, the selection bit for the two register groups, an interrupt inhibit bit, a carry enable bit, a logical/arithmetic compare select bit, and flag and sense bits for external bit-serial interfacing.

It has been said that the 2650 has the most minicomputer-like instruction set of any microprocessor currently available. There are 75 basic instructions, but many of these are actually subdivided into a number of variations. For example the eight arithmetic instructions may be performed either with, or without car-



The basic architecture of the 2650 microprocessor chip showing major data, address and control paths.

ry/borrow; this also applies to the two rotate instructions. Similarly the four compare instructions may perform either arithmetic or logical comparison, while four of the 12 branch instructions and six of the ten subroutine branch/return instructions are conditional upon the two PSW condition code bits—giving typically about 3 possible variants.

Also although there are nominally only six input-output transfer (IOT) instructions, as distinct from the memory reference instructions (which may also be used for IOT), two of these are "extended" instructions which may address any one of 256 distinct 8-bit input-output ports.

One, two and three-byte instructions are used, giving a high degree of programming efficiency. Register-to-register and simple IOT instructions are one byte, extended IOT instructions are two bytes, while memory-reference instructions are either two or three bytes long as required.

The memory reference addressing modes provided by the 2650 are generally agreed to be the most extensive and versatile of any micro-processor currently available. The modes are as follows:

1. Immediate addressing, with the data mask or value in the second byte of the instruction itself.
2. Direct addressing, either absolute or program relative with a displacement range of from −64 to +63.
3. Direct indexed addressing, absolute.
4. Direct indexed addressing with auto increment.
5. Direct indexed addressing with auto decrement.
6. Indirect addressing, either absolute or program relative with a displacement range of from −64 to +63.
7. Indirect addressing with post indexing.
8. Indirect addressing with post indexing and auto increment.
9. Indirect addressing with post indexing and auto decrement.

Memory and IOT interfacing of the 2650 is asynchronous, using "handshaking" control signals. This makes the 2650 compatible with almost any type of memory and peripheral device.

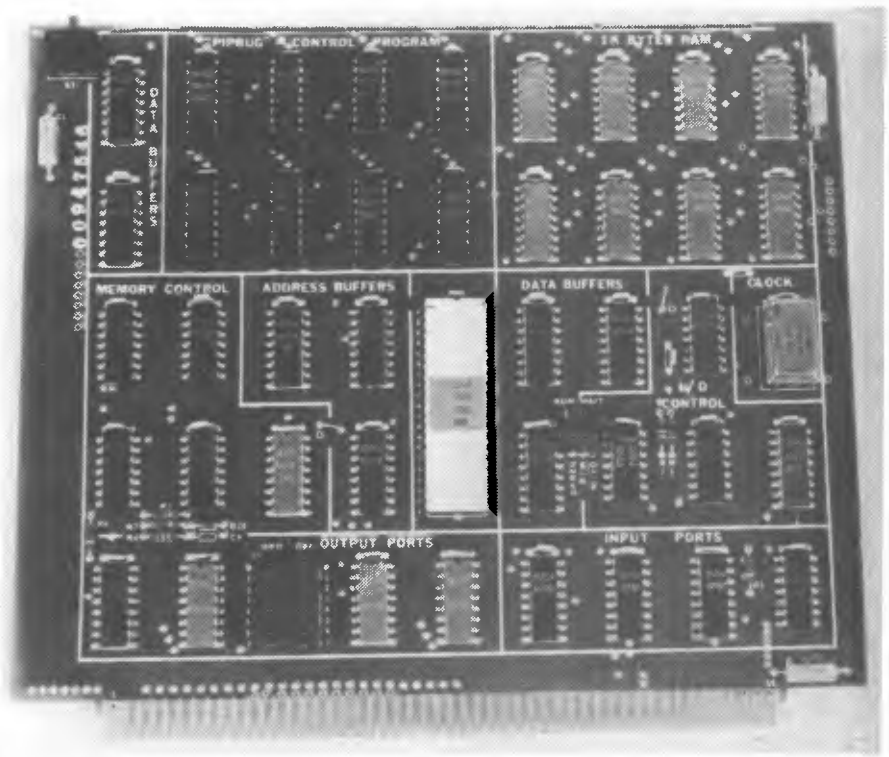
The 2650 has a single level vectored interrupt capability. When it enters the interrupt mode, the chip is able to input an 8-bit address vector via the data bus. This may be used with either direct or indirect addressing to access interrupt servicing routines in any part of the memory space.

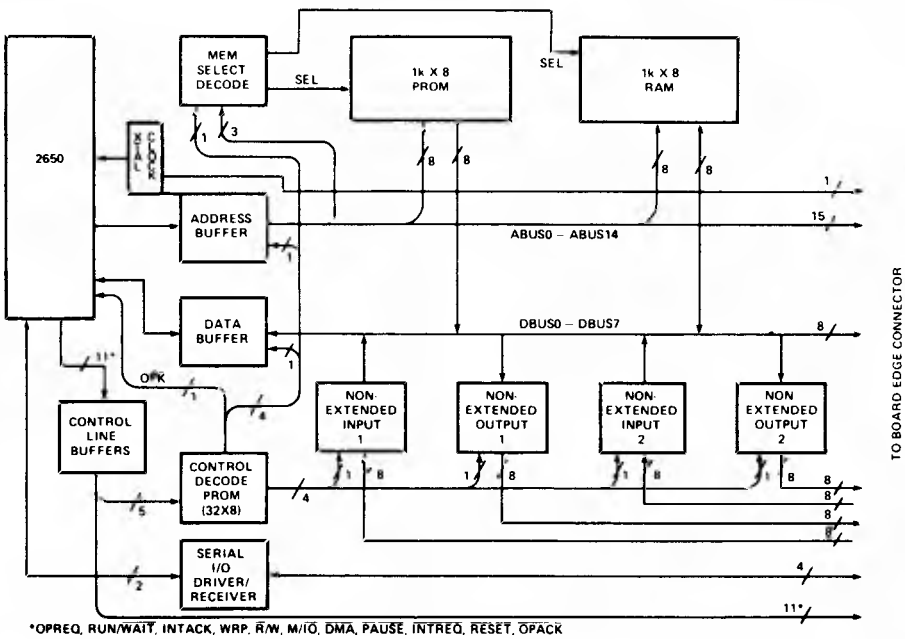
As you should be able to see from this brief rundown of its salient features, the 2650 is a particularly flexible microprocessor, and one which is very well suited for general-purpose microcomputer applications. As such it would seem a good choice for anyone seeking to build up a minicomputer-type system based on a microprocessor.

At the same time, the relatively low cost of the basic chip (currently around $20) and its ability to operate with little more than a clock generator and a ROM in dedicated mode would also make it a good choice for low level applications.

Signetics make two evaluation kits based on the 2650, and these are currently available in Australia from Philips. The more elaborate of the two is the PC1001, which comes as a ready-wired PC board together with edge connector socket and literature. The other kit is a little less elaborate, and comes as either a completely wired PC board or as an assemble-it-yourself kit. In wired form it is designated the PC1500, while the D-I-Y version is the KT9500.

Both kits are basically small microcomputers, capable of being used directly with a power supply and an ASCII





The PC1001 evaluation board system, which is also pictured at top of this page.

teleprinter to develop small 2650 programs in machine language. They could also be expanded into quite pretentious minicomputer systems, by adding further memory and IOT facilities.

An add-on RAM memory board is in fact available, and is directly compatible with either kit. Designated the PC2000, it provides an additional 4k bytes of memory.

At the time of writing this article, we have only had the opportunity to examine and use one of the PC1001 evaluation kits. This is on a PC board measuring 203 × 175mm, with a 100-way double sided edge connector along one of the longer sides. The PC board is pictured, and as you can see there are quite a few IC's apart from the microprocessor.

In fact the PC board is a three-layer assembly, with copper conductors sandwiched in between two layers of epoxy-fibreglass as well as on the two external surfaces. This has allowed Signetics to fit a surprisingly large amount of circuitry on the relatively modest PCB area.

On the PC1001 board is 1k bytes of RAM, capable of storing quite a respectable user program. In addition there is another 1k bytes of ROM, containing a resident monitor-debug program which Signetics have dubbed "PIPBUG". This will be described shortly.

There is an on-board serial asynchronous teleprinter interface, which may be adjusted by means of wire links for either 20mA current loop interfacing or RS232-type voltage interface.

In addition to the teleprinter interface there are four 8-bit parallel IOT ports—two inputs and two outputs. These are wired to be accessed via the two-byte "non-extended" IOT instructions, so that small systems with four or less peripherals (apart from the teleprinter) may be implemented with no further hardware.

The PC1001 board has a 1MHz crystal clock, and therefore is immediately compatible with a 110-baud teleprinter (serial formatting is done by firmware routines, so baud timing is derived from the system clock).

Full data and address bus buffering is provided on the PC1001 board, to simplify addition of further memory or peripherals. All of the control signals are also available at the edge connector in buffered form, which again simplifies any required interfacing.

Although at the time of writing we have not had the opportunity to examine and use the PC1500/KT9500 evaluation kit, we understand that this is based on a PC board identical in size to that of the PC1001. And although the second kit is nominally a less pretentious one, it still offers quite impressive facilities.

For example it still provides 1k bytes

of ROM, with the same resident monitor-debug program (PIPBUG) provided on the PC1001. The only difference in terms of on-board memory is the RAM, which in this case is of only 512 bytes. This is still adequate for a lot of modest programming, of course—and you can always add further memory, as the board again provides fully buffered data, address and control signal buses.

The serial asynchronous teleprinter interface is still provided, but there are now only two 8-bit parallel IOT ports. However, these are programmable in terms of direction, so that they may be used for both input and output.

In place of the crystal clock, the PC1500/KT9500 has an R-C clock oscillator using a 74123 dual monostable.

As not all of the PC board is used by the basic circuitry of the PC1500/KT9500 system, the unused area is provided with plated through holes on 0.3in centres, to allow fitting of additional memory/peripheral decoding ICs if desired.

In short, the PC1500/KT9500 evaluation kit is only a little less flexible than the PC1001. Both are in reality small development systems, capable of being used to develop and run 2650 programs. And in their basic form, each could be used to develop programs for running on the other—apart from the memory size difference. In that sense they are software compatible.

Not only this, of course, but because they use the same resident monitor-debug program they are also virtually identical in the operating sense.

As evaluation kit resident debug programs go, PIPBUG seems quite a flexible one. It recognises seven basic commands, each of which consists of an alphabetic character, any required numerical parameters, and a terminating carriage return. The parameters are given as hexadecimal characters, with leading zeroes unnecessary.

The seven commands and their functions are as follows:

A – See and alter memory
B – Set breakpoint (2 permitted)
C – Clear breakpoint
D – Dump memory to paper tape
G – Go to address, run
L – Load memory from paper tape
S – See and alter registers

The D command may be used to punch out any desired range of memory locations, with leader, checksum and trailer to facilitate reloading. Both the A and S commands may be auto-incremented, by terminating with a line feed instead of a carriage return.

A full listing of PIPBUG is provided with the evaluation kits, which is very useful. Among other things, it allows the user to make use of the teleprinter servicing subroutines in PIPBUG, by arranging for application programs to call them as required.

```
'SIMPLE ANSWER-BACK PROGRAM FOR SIGNETICS PC1001 SYSTEM
DEVELOPED BY J.ROWE,"ELECTRONICS AUSTRALIA" MAGAZINE 11/7/76
NOTE: PROGRAM STARTS AT LOCATION 500 (HEX)

LISTING:

500  76 C0      PPSU     40    /SET UP TTY
502  3F 02 86   BSTA,UN CHIN  /FETCH CHAR FROM TTY VIA PIPBUG RTN
505  C1         STRZ,R1       /SAVE
506  3F 02 B4   BSTA,UN COUT  /ECHO VIA PIPBUG ROUTINE
509  01         LODZ,R1       /RESTORE IN R0
50A  A4 0D      SUBI,R0 "CR"  /R0= CHAR -CR
50C  58 74      BRNR,R0 -12   /CR? IF NOT KEEP GOING
50E  04 0A      LODI,R0 "LF"  /SUPPLY LF
510  3F 02 B4   BSTA,UN COUT
513  05 00      LODI,R1       /SET R1=0
515  0D 25 26   LODA,R1 526+  /FETCH ANSWER CHAR
518  C3         STRZ,R3       /SAVE
519  3F 02 B4   BSTA,UN COUT  /PRINT
51C  A7 0D      SUBI,R3 "CR"  /R3= CHAR -CR
51E  5B 75      BRNR,R3 -11   /CR? IF NOT KEEP GOING
520  04 0A      LODI,R0 "LF"  /YES; SUPPLY LF
522  3F 02 B4   BSTA,UN COUT
525  1B 5A      BCTR,UN -38   /BACK TO LOOK FOR NEW INPUT
527  47 4F 20                 /START OF ANSWER BUFFER
52A  41 57 41
52D  59 2C 49
530  27 4D 20
533  42 55 53
536  59 21 0D                 /ANSWER MUST END WITH CR (HEX 0D)

SAMPLE OF OPERATION:

*G500

HELLO THERE, WHAT AT PRESENT ARE YOU COMPUTING?
GO AWAY,I'M BUSY!

DON'T BE LIKE THAT,PLEASE
GO AWAY,I'M BUSY!
```

This simple novelty program was written largely to verify that the teletype servicing routines in PIPBUG could be called by a user program. The listing shows instruction mnemonics and comments as well as the actual instructions in hexadecimal code.

To illustrate this, the author wrote a simple novelty program for the PC1001 system. Its listing and a sample of the operation are reproduced on these pages, and as you can see it does nothing more than monitor input from the teleprinter, waiting for the person at the keyboard to press the carriage return key. When this occurs, it responds by typing out a curt reply: "GO AWAY, I'M BUSY!".

I wrote this little program mainly for practice with the 2650 instruction set, and also to check out the use of the PIPBUG teleprinter servicing routines. The program inputs characters via the "CHIN" subroutine in PIPBUG, whose calling address is 0286, and outputs characters via the "COUT" subroutine whose calling address is 02B4. As you can see the program itself starts at location 0500.

Note that the program uses one, two and three-byte instructions, and requires only 57 bytes of memory including the answer message buffer. This illustrates the programming efficiency possible with the 2650's powerful instruction set and wealth of memory addressing modes.

If you're interested in the PC1001 or the PC1500/KT9500 evalution kits or the PC2000 add-on memory, they are available from the Electronic Components and Materials Division of Philips Industries, with offices in each state, or from their distributors. Prices for the kits are as follows:

PC1001 – $345 plus tax
PC1500 – $245 plus tax
KT9500 – $165 plus tax
PC2000 – $400 plus tax

Each of the basic kits comes with all of the literature needed to use it. All you need is a power supply and a teleprinter. The teleprinter must communicate in ASCII code, as with most other kits. Here at EA we are currently working on a way to allow this to be done at low cost using a surplus Baudot teleprinter.

The Fairchild F8

This month we continue our survey of microprocessors with a more detailed look at the Fairchild F8 chip set. We also look at Fairchild's F8 Design Evaluation Kit, which comes as a fully assembled PC board complete with power supply and teleprinter cables, together with a set of user, programming and applications manuals.

by JAMIESON ROWE

Fairchild Semiconductor's F8 microprocessor is an 8-bit design, like most of the others we have looked at in previous articles. Current devices are of the familiar N-channel MOS type, made using Fairchild's established "Isoplanar" LSI technology.

Despite these superficial similarities, the F8 microprocessor is significantly different from the others we have examined. Designed primarily for high-volume dedicated-use applications, its basic architecture is quite unlike that of a conventional minicomputer. As a result of this, its instruction set also tends to be rather different.

At present, two chips form the heart of any F8 system. One is the 3850, designated the CPU or central processing unit, and the other is the 3851 program storage unit or PSU. In terms of chip area the latter device is primarily a mask-programmed ROM, organised as 1k bytes, which stores the program to run the system.

This may sound conventional enough, but if you look at a block diagram for any F8 system, it shouldn't take long to realise that there is something missing: the usual address bus.

In fact, the designers of the F8 system have avoided altogether the need for conventional address bus, by moving all of the memory addressing registers out of the CPU chip, and into the PSU.

The 3850 chip is therefore unlike most other CPU chips, in that it contains no program counter, pointer, stack, index or other addressing registers. But on the other hand it does contain 64 bytes of "scratchpad" RAM memory and two 8-bit bidirectional I/O ports—things one doesn't tend to find on other CPU chips! These extra "goodies" have been provided by taking advantage of the chip area and package pins which would otherwise have been used for memory addressing.

On the other hand the mating 3851 PSU device contains a number of things which one doesn't find in a normal ROM, such as a program counter, a stack register, a data counter or indirect memory addressing register, and interrupt control logic. Quite apart from these it also provides two further 8-bit bidirectional I/O ports, and a programmable timer.

Block diagrams of the 3850 and 3851 devices are shown below.

What all this means is that the two chips alone provide a fully viable microcomputer system, with 1k bytes of ROM program storage, 64 bytes of RAM, inbuilt clock and programmable timer, and 32 bits of programmable I/O. This is sufficient for a great many dedicated applications, such as automotive controllers, home appliance controllers, electronic scales and point-of-sale terminals, electronic games and information processors.

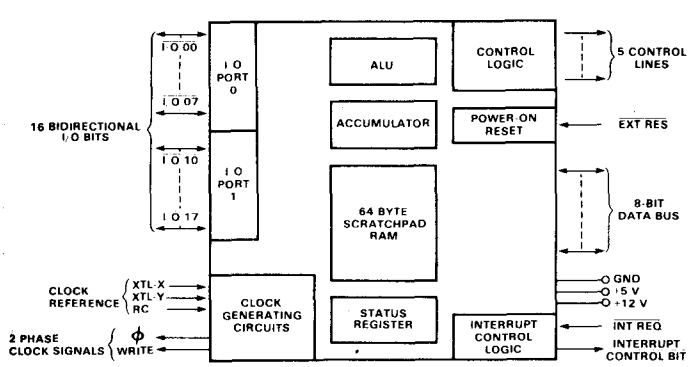
A huge market is predicted in this area, and Fairchild has fairly obviously aimed the F8 system at it in terms of cost-effectiveness. This is evident in the novel architecture adopted for the two basic chips, and also in their recent announcement of a new "single chip F8" to be released early new year. To be called the 3859, the new chip will combine the 3850 CPU and the 3851 PSU in a single package priced at less than $20 in large quantities.

Fairchild is confident that this will give them a big slice of the total microcomputer market. In fact, according to Peter Duddy, F8 marketing engineer for Fairchild Australia, Fairchild expects to be the largest supplier of microprocessor chips in the world, by the end of this fiscal year.

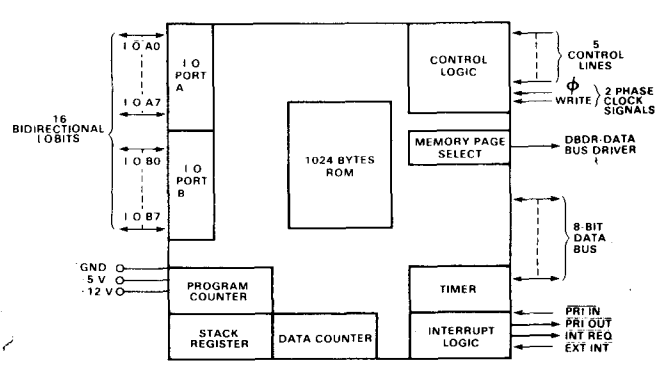
It should perhaps be pointed out that since the ROM in the current 3851 PSU and that in the forthcoming 3859 are both mask-programmed, neither device is really suitable for small-quantity applications. There is a mask charge of around A$1250, and a minimum order quantity of 200 pieces.

Although the F8 system would thus appear to have been designed primarily for dedicated applications requiring a minimal configuration, there is at the same time adequate provision for expansion into more elaborate and memory-intensive systems.

The memory address registers in the PSU chip are 16 bits wide, so that the F8 system is potentially capable of addressing up to 65k bytes of memory. A single PSU chip at present provides only 1k bytes of ROM, although a 3k version has apparently been developed, and should be available shortly. Both are mask-programmable in terms of the actual location they occupy in the overall 65k memory space.



3850
CPU BLOCK DIAGRAM



3851
PSU BLOCK DIAGRAM

This means that multiple PSU chips may be used, to provide whatever amount of program storage is required. Each PSU will have its own program counter and other addressing registers, and all of these will work in tandem. However as the PSU's are mask-programmed to slot into different parts of the overall memory space, only one PSU is ever active during any particular instruction fetch or memory data operation.

RAM memory can of course be added to the system also, although not quite in the usual way. Because there are no address lines available, the RAM chips must be interfaced via special chips which duplicate the program counter and memory addressing registers of a PSU, together with the interfacing control logic.

There are two RAM interfacing chips, one for static memory devices (the 3853), and the other for dynamic memory devices (3852). The latter also provides logic for direct memory interfacing (DMA), in conjunction with a further dedicated DMA chip called the 3854.

There are only three programmable registers in the F8 CPU chip, apart from the 64-byte scratchpad. The three registers comprise an 8-bit primary accumulator, a 6-bit register used for indirect addressing of the scratchpad (called the ISAR), and a 5-bit status register.

To a certain extent the 64-byte scratchpad acts like a bank of 64 secondary 8-bit accumulators. The first 11 scratchpad bytes are directly addressable via some of the F8 instructions, while the rest are accessible through implied addressing via the ISAR register. However, scratchpad addresses 9-15 inclusive (decimal) are dedicated as buffers for the PSU addressing registers, so these will not usually be available for other purposes.

On the software side, the F8 has a repertoire of some 76 instructions, more than half of which use a single byte. Of the rest only three use 3 bytes, and the remainder 2 bytes. This allows some programs to be surprisingly short.

The F8 designers have achieved this economy by relying fairly heavily on implied addressing, where the data to be used in executing an instruction is not specified either directly or indirectly via an instruction operand, but is simply implied by the type of instruction. Most microprocessors use this addressing mode for arithmetic and logic instructions, for example, which normally "imply" that the data concerned is the content of the accumulator.

The F8 carries this considerably further. All eight of the non-branch memory-reference instructions use implied addressing, implying the PSU data counter as an indirect address register. Similarly all 15 of the scratchpad register instructions use implied addressing, implying either the accumulator or the use of the ISAR as an indirect scratchpad address register, or both. Quite a number of other instructions also use implied addressing.

In all there are some 15 accumulator instructions, 12 branch instructions, 8 memory reference instructions, 13 address register instructions (including jump to subroutine and return), 15 scratchpad register instructions, and 13 miscellaneous instructions.

Incidentally all memory reference instructions involve automatic post-incrementing of the PSU data counter, which can be very useful. Many of the scratchpad instructions have optional auto increment or decrement of the lower 3 bits of the ISAR implied indirect addressing register, which again can be useful.

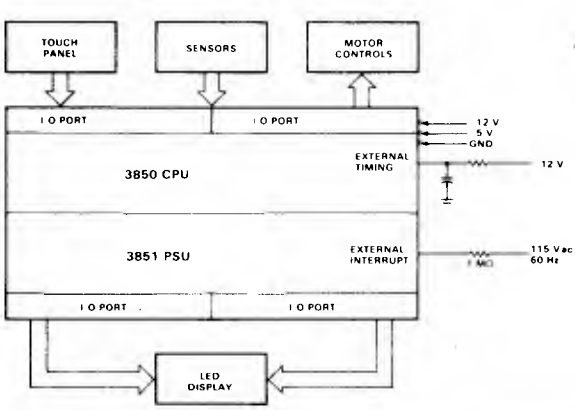
Included in the F8 instruction set are a number of powerful immediate instructions, including ADD, AND, COMPARE (2's complement subtraction), EX-OR, LOAD and OR, together with a "call to subroutine immediate" and a "load data counter immediate". There is also a "short" LOAD immediate instruction, which is only a single byte long, and used to load the accumulator with 4-bit data.

Input/output servicing is normally handled by two separate 2-byte instructions, INPUT and OUTPUT, both of which use the second byte to address one of the IOT ports provided by the CPU chip and the PSU chip or chips. However there are also two "short" IOT instructions, capable of transferring data to and from the lowest 16 IOT ports.

Let us now pass from the F8 chips and their operation to look at the "F8 Design Evaluation Kit" currently available from Fairchild, to allow potential users to get practical experience and undertake simple program development at low cost.

As you can see from the picture, the

As received, the Fairchild F8 evaluation kit comprises an assembled PC board system, an edge connector with cables, and a binder of manuals.



Together the F8 CPU and PSU chips are capable of forming a complete minimal system, suitable for appliance controllers and similar applications.

kit comes in three parts. There is a fully assembled PC board containing the evaluation system itself, an edge connector fitted with power supply and teleprinter cables, and a large ring binder containing a user manual, programming manuals and various other pieces of useful literature.

The evaluation system on the PCB contains a 3850 CPU, a 3851 PSU and a 3853 static memory interface to which is connected 1k bytes of static RAM. Three of the four IOT ports provided on the 3850 and 3851 chips are available for interfacing to 8-bit parallel peripherals, while the fourth is dedicated to serial interfacing with either a 100-baud teleprinter or a 300-baud terminal, as desired.

The 3850 clock oscillator is implemented using RC components, which must be "tweaked" to give a clock frequency of 2MHz using a CRO or digital counter—not for the F8 system itself, but to ensure the correct baud rates for the serial interfacing.

The edge connector cables provided with the board have banana-type plugs to connect to a power supply, and a Molex 15-way connector which is capable of directly into the "No. 2" socket on the rear right of a standard model ASR-33 or KSR-33 Teletype. The power supply requirements are for +5V at 500mA, and +12V at 100mA.

Resident in the ROM section of the PSU is Fairchild's debug program, which they have dubbed "FAIR-BUG". As with similar programs provided in the other evaluation kits we have examined, this provides basic facilities for the user to develop programs, together with routines for teleprinter IOT servicing. There is also a routine to input a byte of data from one of the parallel IOT ports, something not usually found. All of the IOT routines are available to be called by user programs as subroutines.

The command set recognised by FAIR-BUG permits the user to display and optionally alter memory locations, scratchpad registers, the accumulator, ISAR and status registers or the PSU addressing registers. It also allows the user to punch and load formatted paper tapes, and to execute the user's program. As well as these fairly common functions there is another useful function which is not often provided: a command to punch out the user's program in PROM burning format, as distinct from reloadable format.

Fairchild Australia very kindly made one of the F8 evaluation kits available, so that we could gain some first-hand knowledge and "hands-on" experience. The hardware side of the system turned out to be quite easy to hook up, although I had to remember to adjust the 3850 clock oscillator to 2MHz using a digital counter—to ensure that the timing would be right for the teleprinter interface. With this done the system was soon up and running.

Of course learning to drive a microprocessor's instruction set takes some time, as does the business of becoming fully familiar with the command repertoire of a debugging program. Both of these seemed to present a little more of a challenge with the F8 system than with the other systems I have used, perhaps because both the F8 instruction set and the FAIR-BUG commands are structured somewhat differently. However, after digesting the various manuals and accompanying literature, I was soon doing some tentative program development.

Like some of the other small debugging programs, FAIR-BUG does tend to be rather wasteful in terms of teleprinter paper, by insisting on the use of a carriage return as a command terminator. One tends to use metres of paper, but with characters printed mainly in a narrow column at the left-hand side. The method used to initially keyboard in a program in hexadecimal code using FAIR-BUG also seems a little clumsy, requiring the use of two separate commands per entry.

However, while mildly irritating, neither of these could be construed as major shortcomings.

Out of interest, I tried writing a simple "answer-back" program like those I wrote for the systems we have discussed previously in this survey. By no stretch of the imagination is this a critical "benchmark" program designed to show up the relative capabilities of systems. However, it does provide a modest basis for comparison.

A listing of the program is shown on these pages, and you may find it of interest. As before it uses the teleprinter servicing subroutines in the debug ROM, which are here labelled "TTYI" and "TTYO" respectively. The program calls them by means of immediate addressing "jump to subroutine" instructions, which have the mnemonic "PI".

Note that because the TTYI subroutine does not appear to strip off the incoming parity bit, the exclusive-OR immediate instruction ("XI") used to test for an incoming carriage return tests for a code of 8D hexadecimal, not the correct ASCII code of 0D. Note also that the "FCH" loop used to fetch and deliver the answer does not need to increment a buffer pointer, because the memory reference instruction "OM" causes this to happen automatically. The OM (OR from memory) is used here in preference to an "LM" (load from memory) instruction, because the LM instruction does not affect the status register, and the following BZ (branch on zero) instruction is determined by the status register rather than the actual accumulator content.

All things considered, I found the Fairchild evaluation kit fairly easy to drive, and one which provides a practical and low cost way of becoming familiar with the F8 microprocessor system. It should be of particular interest to those intending to design with the F8, either as an evaluation system or as a low-cost development tool.

Price of the kit is quoted as $166.50, plus tax where applicable. Kits are available from authorised Fairchild distributors in each state.

```
ANSWER-BACK PROGRAM FOR FAIRCHILD F8 EVALUATION KIT
J.ROWE, ELECTRONICS AUSTRALIA 9/9/76

0000 28 83 AD  GO,PI   83AD     /CALL TTYI SUBROUTINE
0003 53           LR   3,A      /STORE CHAR IN R3
0004 28 83 E5     PI   83E5     /CALL TTYO TO ECHO
0007 43           LR   A,3      /RETURN CHAR TO AC
0008 23 8D        XI   '8D'     /CR?
000A 94 F5        BNZ  GO       /LOOP BACK & CONTINUE IF NOT
000C 7A           LIS  '0A'     /YES; LOAD LF
000D 51           LR   1,A      /STORE IN R1
000E 28 83 E5     PI   83E5     /SEND TO TTY VIA TTYO SUBR.
0011 2A 00 1F     DCI  001F     /SET DC TO START OF ANSWER BUFF
0014 70        FCH,CLR          /CLEAR AC
0015 8B           OM           /OR ANSWER CHAR INTO AC, CHECK STATUS
0016 84 E9        BZ   GO       /RETURN TO START IF CHAR IS ZERO
0018 51           LR   1,A      /STORE CHAR IN R1
0019 28 83 E5     PI   83E5     /SEND TO TTY VIA TTYO
001C 90 F7        BR   FCH      /KEEP GOING UNTIL FINISHED
001E 00
001F 47 4F 20                   /START OF ANSWER BUFFER
0022 41 57 41
0025 59 2C 49
0028 27 4D 20
002B 42 55 53
002E 59 21 0D
0031 0A 00                      /ANSWER MUST END WITH A ZERO BYTE
```

Here is the author's novelty answer-back program as implemented for the F8 evaluation kit. It illustrates some of the points discussed.

Microprogramming module

For those who need to know the "inner mysteries" of microprocessors, including the way they execute instructions, Texas Instruments are producing a series of learning modules. The first of these to be released is the LCM-1001 Microprogrammer, designed to demonstrate the concepts of microprogramming.

by JAMIESON ROWE

Before describing the LCM-1001, it should perhaps be stressed that it is definitely NOT a microprocessor evaluation kit or development system. It is basically a teaching aid, designed primarily to demonstrate specific details of a microprocessor chip's internal operation, and perhaps to serve as a tool for the designers of specialised complex systems.

When the other modules in the TI series become available, it will apparently be possible to interconnect the LCM-1001 with them to produce both a functioning microprocessor, and ultimately a complete microcomputer system. But the resulting systems will be "micro" only in the sense that appropriate LSI chips will be buried within the various modules.

The LCM-1001 module itself is based on a 40-pin LSI device called the SBP-0400, which Texas Instruments describe as a "4-bit expandable parallel binary processor element". It appears to be essentially a 16-function 4-bit arithmetic and logic element (ALU) combined with a register file of eight 4-bit registers, two 4-bit working registers, and a factory programmed logic array capable of controlling the functions of these sections in response to 9-bit instruction words.

The prime use of the SBP 0400 element is in making processors based on "bit-slice" architecture. It provides a 4-bit wide slice of the main part of a processor, so that a number of units may be "stacked" to produce processors of any desired size—8 bits, 12 bits, 16 bits or as large as necessary.

To produce a complete microprocessor, the group of SBP 0400 devices must be supplemented by an algorithm controller. This is because the SBP 0400 is a microprogrammable device, capable of doing only one elementary task at a time. But it may be programmed to do any of the 512 tasks capable of being encoded by its 9-bit instruction word, so that with a suitable algorithm controller a group of SBP 0400 devices form a very flexible processor.

What the algorithm controller does is take the operation code portion of the "machine language" instruction words for the processor, and produce whatever is the correct sequence of elementary "microinstructions" necessary for the SBP 0400 devices to perform the required task. This is the technique of microprogramming—there is virtually a "computer within the computer", or more strictly a controller within the processor.

Many modern microprocessors use the microprogramming technique, which has the advantage that the instruction repertoire of the processor may be changed simply by changing the micro-instruction sequences stored in the algorithm controller (usually in a ROM). With processors that are not microprogrammed the instruction repertoire generally cannot be changed without extensive and costly redesign of the processor chip as a whole.

It is basically the technique of microprogramming that the LCM-1001 module is designed to demonstrate, using the SBP 0400 device as an example. Each of the control inputs, data inputs and data outputs of the device are brought out the LED indicators, and all inputs are provided with miniature toggle switches so that the device may be manually programmed. A pushbutton provides the clock signal input.

In addition, all of the SBP 0400 pin connections are brought out to a 40-pin DIL socket for possible future interconnection to other modules. The power for the module comes from an internal rechargeable battery, with a simple rectifier circuit connected to a miniature jack socket to allow for battery charging. An external transformer unit is supplied with the module to provide the required 5.6V AC, but as this unit is designed to plug directly into a 110V outlet, it will not be of much value to Australian buyers.

The manual supplied with the LCM-1001 module is very comprehensive, although it does assume that the reader has a sound grasp of basic microprocessor operation and programming.

All in all, the Texas Instruments LCM-1001 learning module would seem a very effective way of demonstrating the concepts of processor microprogramming. It should find considerable use in universities and colleges, and also in design labs engaged in the development of custom microprogrammed processor systems.

Of course it isn't really necessary to know how a microprocessor executes its instructions in order to use it or program it. So that the majority of people working with microprocessors and microcomputers are unlikely to need the detailed knowledge that the LCM-1001 is designed to teach and demonstrate. In that sense it must be seen as a rather specialised learning tool.

Further information on the module and its planned companions is available from the local distributors for Texas Instruments, Instant Components Service, who have offices in Sydney, Melbourne and Adelaide.

The Texas Instruments LCM-1001 microprogramming learning module, complete with its construction manual. Later modules will deal with other aspects of processors and computer systems.

The Mostek F8

Our survey of microprocessors continues this month with a detailed look at the Mostek F8 chip set, and the Mostek Evaluation Kit. This comes either as a fully assembled printed circuit board, or as a kit of parts.

by DAVID EDWARDS

The Mostek Corporation is second sourcing the Fairchild F8 microprocessor chip set, which was featured in last month's article. As noted in that article, the F8 microprocessor set is significantly different from other 8 bit designs we have considered. It is manufactured using N-channel Isoplanar MOS technology.

The F8 system is designed primarily for high-volume dedicated-use applications, and does not lend itself quite as readily as some other systems to one-off designs. However, this system can function with only two chips as a complete, fully viable microcomputer system, with 1k bytes of ROM program storage, 64 bytes of RAM, inbuilt clock and programmable timer, and 32 bits of programmable I/O.

The two chips which form the heart of the F8 system are the 3850, designated the CPU or central processing unit, and the 3851, program storage unit or PSU. In terms of chip area the latter device is primarily a mask-programmed ROM, organised as 1k bytes, which stores the program to run the system.

Conventional bus addressing has been avoided by moving all the memory address registers out of the CPU and into the PSU. The extra chip area so gained on the CPU chip has been used to incorporate a 64 byte scratchpad RAM and two bidirectional I/O ports.

On the other hand the mating 3851 PSU device contains a number of things which one doesn't find in a normal ROM, such as a program counter, a stack register, a data counter or indirect memory addressing register, and interrupt control logic. Quite apart from these it also provides two further 8-bit bidirectional I/O ports, and a programmable timer.

Fig. 1 shows how the basic two-chip F8 system is implemented. Fig. 2 shows a more advanced system, which has further features such as direct memory addressing (DMA). This allows peripheral devices, such as data inputs and outputs, direct access to the computer memory. This is achieved without any degradation in system performance, and has the advantage of using fewer machine cycles than would otherwise be the case.

There are only three programmable registers in the F8 CPU chip, apart from the 64-byte scratchpad. The three registers comprise an 8-bit primary accumulator, a 6-bit register used for indirect addressing of the scratchpad (called the ISAR), and a 5-bit status register.

To a certain extent the 64-byte scratchpad acts like a bank of 64 secondary 8-bit accumulators. The first 11 scratchpad bytes are directly addressable via some of the F8 instructions, while the rest are accessible through implied addressing via the ISAR register. However, scratchpad addresses 9-15 inclusive (decimal) are dedicated as buffers for the PSU addressing registers, so these will not usually be available for other purposes.

On the software side, the F8 has a repertoire of some 76 instruction, more than half of which use a single byte. Of the rest only three use 3 bytes, and the remainder 2 bytes. This allows some programs to be surprisingly short.

The F8 designers have achieved this economy by relying fairly heavily on implied addressing, where the data to be used in executing an instruction is not specified either directly or indirectly via an instruction operand, but is simply implied by the type of instruction.

In all there are some 15 accumulator instructions, 12 branch instructions, 8 memory reference instructions, 13 address register instructions (including jump to subroutine and return), 15 scratchpad register instructions, and 13 miscellaneous instructions.

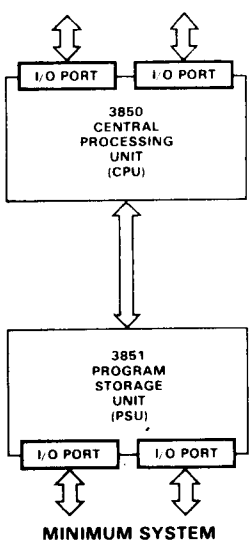
Included in the F8 instruction set are a number of powerful immediate instructions, including ADD, AND, COMPARE (2's complement subtraction), EX-OR, LOAD and OR, together with a "call to subroutine immediate" and a "load data counter immediate". There is also a "short" LOAD immediate instruction, which is only a single byte long, and used to load the accumulator with 4-bit data.

Readers interested in learning more about the F8 chip-set are referred to the previous article in this series, which appeared in the November 1976 issue.

The Mostek F8 Evaluation Kit is available in Australia from Namco Electronics. Like the other evaluation kits we have looked at, it is intended to allow potential users to gain practical experience and undertake simple program development at low cost.

The kit comes well packaged in a strong cardboard box, and comprises a ring binder containing device specifications and programming manuals. The binder also includes the completed circuit board, stapled inside a conductive package. A separate additional programmer's guide is also included.

The PCB, which can be plugged into an edge connector, contains the CPU, PSU and SMI (static memory interface)



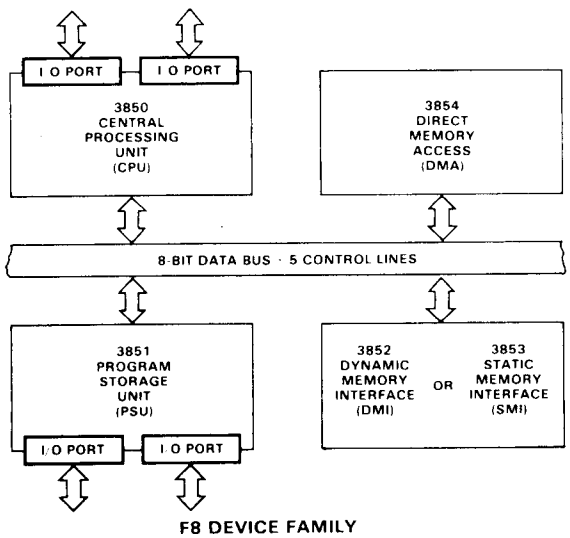


Fig. 1 at left shows a minimal 2-chip F8 system, while Fig. 2 at right shows a more elaborate system with further RAM and DMA facilities.

chips, as well as 1k of RAM, and a full duplex 20mA loop teleprinter interface. A 2MHz crystal is included, so no adjustments are required prior to switch-on. Power requirements are 12VDC at 120mA and 5VDC at 750mA.

The Mostek debug program, called the Designers Development Tool (DDT-1) is resident in the PSU, and occupies the first 1k of memory space. The 1k of RAM appears in the second 1k of memory space, which because of simplified addressing logic also becomes duplicated in the succeeding 1k bytes.

DDT-1 provides basic facilities for development of user programs, as well as routines for teleprinter IOT servicing. DDT-1 allows the operator to insert a breakpoint in his or her program, copy a block of memory into another area, dump memory into and load from paper tape, execute a program starting at any desired memory address, carry out hexadecimal arithmetic, examine and alter memory locations, examine and modify the I/O ports, and to type out a block of memory.

Namco Electronics kindly made available to us a sample evaluation kit, which we were able to "fire up" and try using a Lear Siegler ADM-3 Video Terminal, which was also on hand at the time for review.

DDT-1 proved to be quite easy to use, although, in common with most other simple debug programs, it uses a carriage return as a command terminator. This caused no problems on the video-terminal, but it does tend to cause a teleprinter to consume a large amount of paper. Our only real criticism is that DDT-1 does not seem to be protected against erroneous keyboard entries.

This is the Mostek F8 evaluation kit, comprising PCB, a binder with user manuals, and a programmer's guide. The CPU clock is crystal controlled.

For example, accidental type-in of the letter O instead of a zero when feeding in a hexadecimal address appears to produce unpredictable results. Sometimes part of the user's program can be altered, which can be rather annoying! It would be desirable for DDT-1 to be modified so that it checks for valid hexadecimal characters in the keyboard input, and throws out invalid characters with a query.

Getting to grips with programming proved to be quite straightforward, as Mostek have provided a fully explained sample program. For comparison with other systems, I have written a version of Jim Rowe's "answer-back" program, which is reproduced on these pages.

The first point of interest about this program is the amount of initialization that is required at the start, mainly to use the IOT routines resident in DDT-1. Note also the use of the auto-incrementing ADD BINARY (AM) instruction together with the branch-on-zero (BZ) instruction in the ANSWER loop, to provide exit from the loop at the end without the need for a separate pointer.

Since the DDT-1 teleprinter output routine always responds with a line feed as well as a carriage return when a carriage return is sent, no line feeds are needed in the answer.

Overall, we found the Mostek F8 Kit to be easy to use, and would recommend it to those interested in using the F8 system. It is suitable as either an evaluation system, or as a low cost development tool.

The D.I.Y. kit is available for $248.34 plus tax, while the completely assembled version sells for $300.00 plus tax. Enquiries should be directed to Namco Electronics, 239 Bay Street, North Brighton, Victoria 3186, or to Namco Electronics, 69 Archer Street, Chatswood, NSW 2067.

```
ANSWER-BACK PROGRAM FOR MOSTEK F8 EVALUATION KIT
D. EDWARDS, ELECTRONICS AUSTRALIA 19/10/76

0400 20 FF       INIT,LI   FF          /LOAD AC WITH FF
0402 0B             LR     IS,A        /INITIALIZE ISAR TO 3F
0403 54             LR     4,A         /COPY AC INTO REG 4
0404 34             DS     4           /DECREMENT REG 4 TO FE
0405 56             LR     6,A         /COPY AC INTO REG 6
0406 71             LIS    H'1'        /LOAD AC WITH 01
0407 B6             OUTS   6           /TRANSFER AC TO TIMER PORT TO ENABLE EXT INT
0408 1B             EI                 /ENABLE I/O ROUTINES
0409 2C 03 F3  START,PI    03F3        /CALL TTYIN SUBROUTINE
040C 4C             LR     AS          /COPY  CHAR INTO AC FROM RS
040D 25 0D          CI     'OD'        /COMPARE WITH CR
040F 84 06          BZ     'MESSAGE'   /JUMP TO MESSAGE IF CR
0411 28 03 5D       PI     035D        /SEND CHAR TO TTYOUT SUBROUTINE
0414 90 F4          BR     START       /LOOP BACK TO START
0416 2A 04 23 MESSAGE,DCI  0423        /LOAD DC WITH MESSAGE ADDRESS
0419 70      ANSWER,CLR                /CLEAR AC
041A 88             AM                 /ADD CHAR TO AC AND INC DC
041B 84 ED          BZ     START       /LOOP BACK TO START
041D 5C             LR     S,A         /COPY CHAR INTO RS
041E 28 03 5D       PI     035D        /SEND CHAR TO TTYOUT SUBROUTINE
0421 90 F7          BR     ANSWER      /LOOP BACK TO ANSWER
0423 0D 47 4F                          /START OF ANSWER BUFFER
0426 20 41 57
0429 41 59 2C
042C 20 49 27
043F 4D 20 42
0432 55 53 59
0435 21 0D 00                          /ANSWER MUST END WITH A ZERO BYTE
```

Here is the author's version of our "answer-back" program, re-written for the Mostek evaluation kit and the DDT-1 teleprinter subroutines.

The Lear Siegler ADM-3 video terminal kit

A few months ago in the US, Lear Siegler released an assemble-it-yourself kit version of their ADM-3 video terminal. The kit has now also been released locally by AWA, and although quite a basic "glass teleprinter", it offers the main advantages of a video terminal at a price which should make it of interest to many people working with micro-computers.

by JAMIESON ROWE

Compared with mechanical data terminals like teleprinters, video terminals offer a number of important advantages. They are virtually silent in operation, and the lack of moving parts tends to make them significantly more reliable. They are also potentially capable of communicating with a computer system at much higher rates, being free from the limitations of mechanical strobing and decoding devices.

Of course until a couple of years ago, one had to pay quite a deal of money in order to get these advantages. Video terminals were even more expensive than teleprinters, so that the latter still tended to be the choice in cost-sensitive applications. Happily this all began to change when galloping IC technology entered the MSI-LSI-microprocessor era.

The US firm Lear Siegler Inc. has been quite active in this area, and when it was released their ADM-3 terminal set a new low level for basic video terminal prices. It currently sells for less than the cost of a new teleprinter, even in fully assembled form.

To be sure, the ADM-3 is a fairly basic terminal, compared with some of the new "intelligent" terminals. It doesn't offer inbuilt text editing, page formatting or fancy graphic plotting facilities. Even Lear Siegler describe it as their "dumb terminal", emphasising that it is designed as a teleprinter replacement.

The truth is that a basic terminal of this type is not only quite adequate, but very appropriate for communication with modern micro- and mini-computer systems. Offering quiet, reliable and fast operation at a price significantly lower than most other terminals, it has a lot in its favour.

Now that the ADM-3 terminal is available in kit form for a lower price again, it should even begin to come down into the computer hobbyist realm.

What does the ADM-3 terminal give you? Well, first of all it gives you a 30cm rectangular CRT with P4 phosphor and bonded non-glare faceplate. On this screen can be displayed 960 characters, in 12 lines of 80 characters each. An option gives 1920 characters, in 24 lines.

The display uses the standard 64 6-bit ASCII character set, displayed in 5 x 7 dot matrix form. The cursor homes to the lower left of the screen, and line feeds cause upward stepping of the display with top of screen overflow.

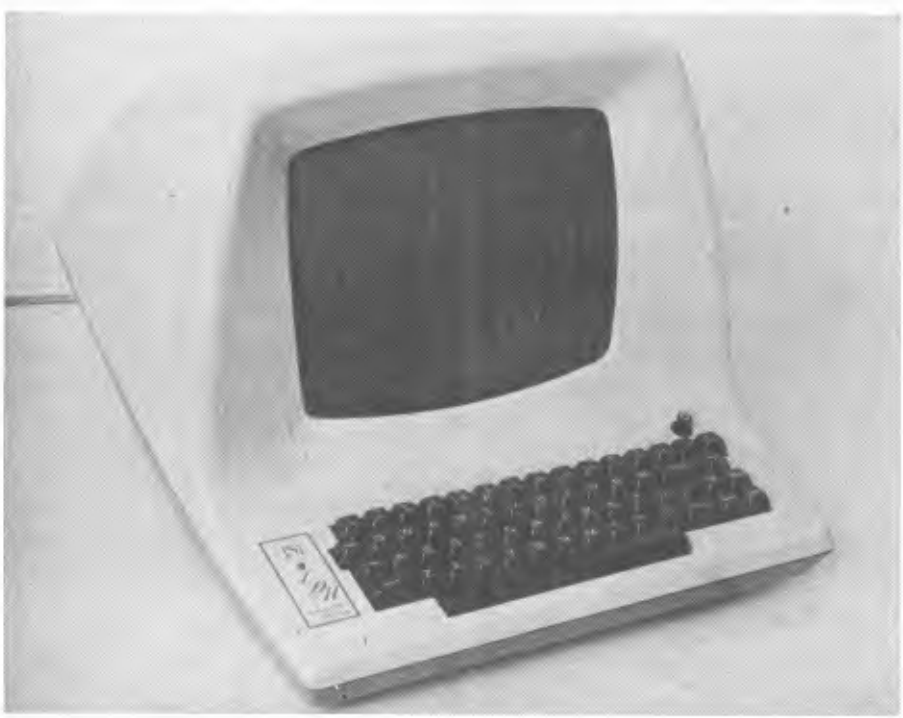
The keyboard fitted to the standard ADM-3 is a standard communications type, with 59 keys. A 10-key numeric pad is available as an optional extra.

Both 20mA current loop and RS232C voltage interfacing facilities are available, with 11 switch selectable communication rates from 75 to 19,200 baud. The terminal may be operated in either full or half duplex modes, and with any of the usual asynchronous data formats.

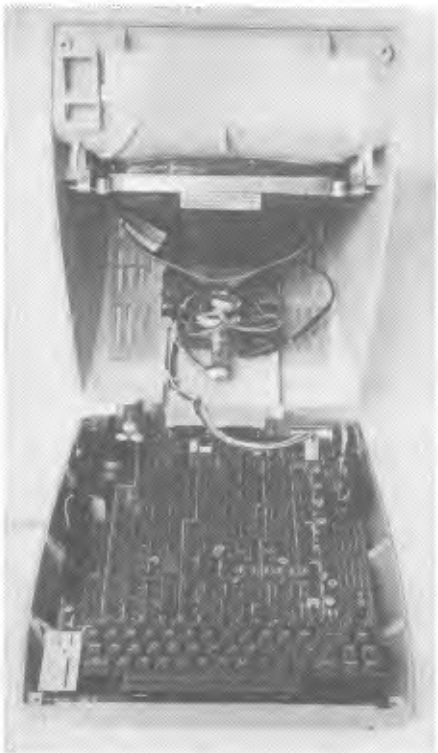
The various data format and communication rate options are set by means of small slider switches accessible beneath a dress plate on the front of the terminal, to the left of the keyboard. This makes it possible to set up the unit without opening the case.

Inside the ADM-3 case virtually all of the internal circuitry is mounted on a large PCB. The only exceptions are the CRT sweep and EHT components, which are in a small assembly above the tube yoke. The main PCB mounts the keyboard as well as the power supply, which should ensure a high standard of overall reliability.

Quite distinct from the front panel communication mode and rate switches, there are internal switches to control special non-printing functions. These



An outside view of the ADM-3 terminal showing its clean, functional styling.



Inside the ADM-3 most of the circuitry is mounted on a single large PCB.

include keyboard locking, screen clearing, and destructive space character.

Thanks to the kind co-operation of AWA Data Systems, we were able to hook up a sample ADM-3 terminal to one of the small microprocessor evaluation systems we have been testing. We found it very easy to set up for 110-baud 20mA interfacing, using the supplied user booklet. And when in operation with the system, we found it very convenient to drive.

Character size is quite reasonable, and the display is well focussed providing one doesn't try for maximum contrast. It takes a while to get used to the line spacing, which is rather wide, but if anything this helps readability.

Even though the ADM-3 has no cooling fan, we found it to run quite cool even after many hours of operation.

In short, then, we found the ADM-3 to be a well-made and thoroughly professional video terminal, of the basic variety. One whose interfacing flexibility and price should make it very suitable for a wide variety of microcomputer system applications.

Price of the fully assembled ADM-3 is quoted at $1395, plus tax where applicable. The assemble-it-yourself kit is quoted as $965 plus tax.

Enquiries regarding the ADM-3 and other Lear Siegler terminals may be directed to AWA Data Systems, 422 Lane Cove Road, North Ryde, NSW 2113. (Telephone 888 8111.)

Microcomputer News & Products

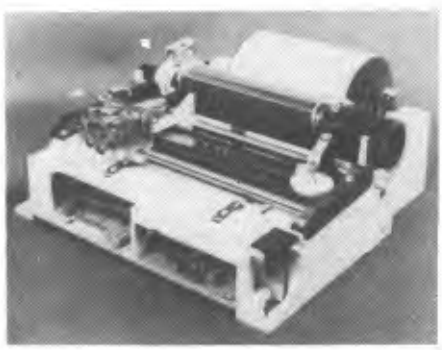
Intel get 3 minutes of music from 2k PROM

Visitors to a recent seminar in Newcastle organised by Warburton Franki, the new Intel Agent and Distributor, were treated to the first performance in Australia of a Bach fugue scored for two Intel 8708 PROMS.

One of the 1k x 8bit memories held the various functions necessary to generate the notes while the other accessed them. The fugue was played on a Prompt 80 development system in full stereo. The performance lasted for a full three minutes, although a continuous encore mode was available.

According to Col Edwards, the Intel Microcomputer Customer Training Manager, Intel spent 3 man-years developing the programs. It seems unlikely then that the popular cassette and record are to be overtaken just yet!

New Philips printer



A new alphanumeric printer unit which runs from a single 12V DC supply has been released by Philips Electronic Components and Materials. Called the 115DR, the new printer operates at up to 66 characters per second and will print lines of up to 40 characters wide on either paper rolls or cards up to 115mm wide.

The 115DR is a mosaic printer, and uses standard typewriter ribbon. It is capable of printing any character which can be formed within a 5 x 7 matrix. In addition, it can be used for facsimile reproduction by driving the printer head in appropriate fashion.

The single 12V power supply, low profile (100mm) and light weight (3kg) make the 115DR suitable for mobile data and communication terminal use as well as for fixed applications. In fact Philips are understandably confident that the 115DR will find application in a very wide range of areas, including data terminals, instrumentation, point-of-sale terminals, police communications, doctors and ambulances.

The features of the 115DR should also make it of interest to computer hobbyists. With a single 12V supply requirement and the ability to reproduce graphics as well as alphanumeric characters, it would make a very attractive terminal for a hobby computer system.

One-off price of the 115DR printer is $338.39 plus tax if applicable. For this you get the printer with head and ribbon system, paper roller and feed mechanism, and motor control electronics. Character generation and solenoid driver electronics are not supplied.

Computers in schools

As part of an innovation project supported by the Schools Commission, Essendon Grammar School in Victoria is producing a series of regular newsletters designed to promote and assist the use of computers in secondary schools. Called COM-3, the newsletter is being edited by Timothy Mowchanuk and Greg Johnstone. Subscription cost is $3 for six issues.

The first issue of COM-3 has articles on hardware and software, listings of game programs, and general news items. The printing is not wonderful, but the editors explain that there were problems which have now been overcome. Later issues should be much more presentable.

Subscriptions should be sent to the editors at P.O. Box 138, Essendon 3040.

An ideal microcomputer for the beginner:

"Mini Scamp"

Forget about expensive terminals: here's a REALLY low cost and simple microcomputer. It uses front-panel bit switches and LEDs for input and output, in normal binary code, making it completely self contained. Based on the National SC/MP microprocessor, it comes with a minimum of 256 words of RAM—but this is easily expanded up to 1024 words. We think it's the ideal way of getting into the exciting world of microcomputers at low cost.

by DR. JOHN KENNEWELL Physics Dept., Newcastle University

The design of this microcomputer started around October of last year with the formation of the Newcastle Microcomputer Club. It became obvious at the inaugural meeting that there were many people who would like to play with their own microcomputer, developing programming skills, yet who were unable to afford even the lowest cost kits available on the market. The problem becomes even more acute when considering the cost of a terminal to interface with these kits.

Various solutions to the terminal problem have now been presented in this magazine. Jim Rowe has described an ASCII-Baudot translator for use with surplus Baudot teleprinter machines (EA, October 1976) and also a video data terminal (EA, January and February 1977). However, either of these alternatives means an outlay of at least about $200, not including the microcomputer itself, which brings the total cost to around $300 using a small system such as the SC/MP evaluation kit.

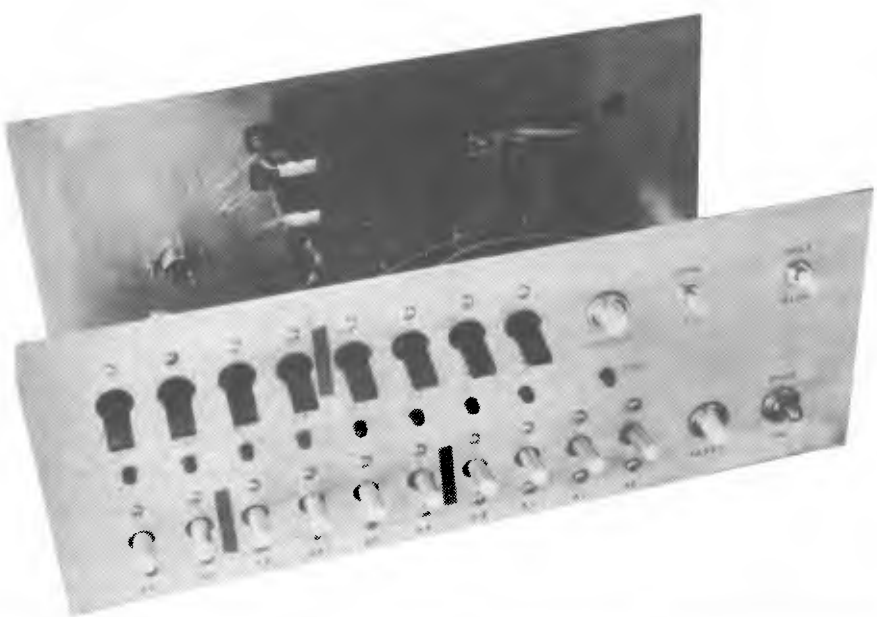
Recently Applied Technology have released a SC/MP I/O kit, which interfaces with the SC/MP evaluation kit and permits program and data entry via panel switches. While this unit undoubtedly fills a gap in available I/O hardware and seems to be enjoying great popularity, the total system cost is over $150 (including power supply). I also feel that it is not suitable for a beginner to microcomputers because it employs an operating system (in ROM) that was designed for communication with a teletype using hexadecimal characters in ASCII code. Entry of information via the panel switches thus involves a prior translation of characters into this code, and one tends to lose contact with the basic organisation of the processor (CPU). From the point of acquiring an understanding of microprocessor operation (without the complications of an intervening operating system) I feel this is undesirable.

In searching for a suitable design, and to overcome the problems mentioned above, it became apparent to me that the idea of using an available evaluation kit together with an I/O interface was not the way to go. As an operating system was not desirable, there was no need for read only memory (ROM). Building a system from scratch meant that costs could be kept down as only those features necessary were included. At the same time, I personally wished to build a much larger system than that shown here, and ease of system expansion was well to the fore in my design considerations. The TOTAL cost of the computer should fall somewhere between $50-$100 depending upon your method of construction, selection of components, and upon how many existing components you have that may be pressed into service. This is most likely to be so in the case of the power supply.

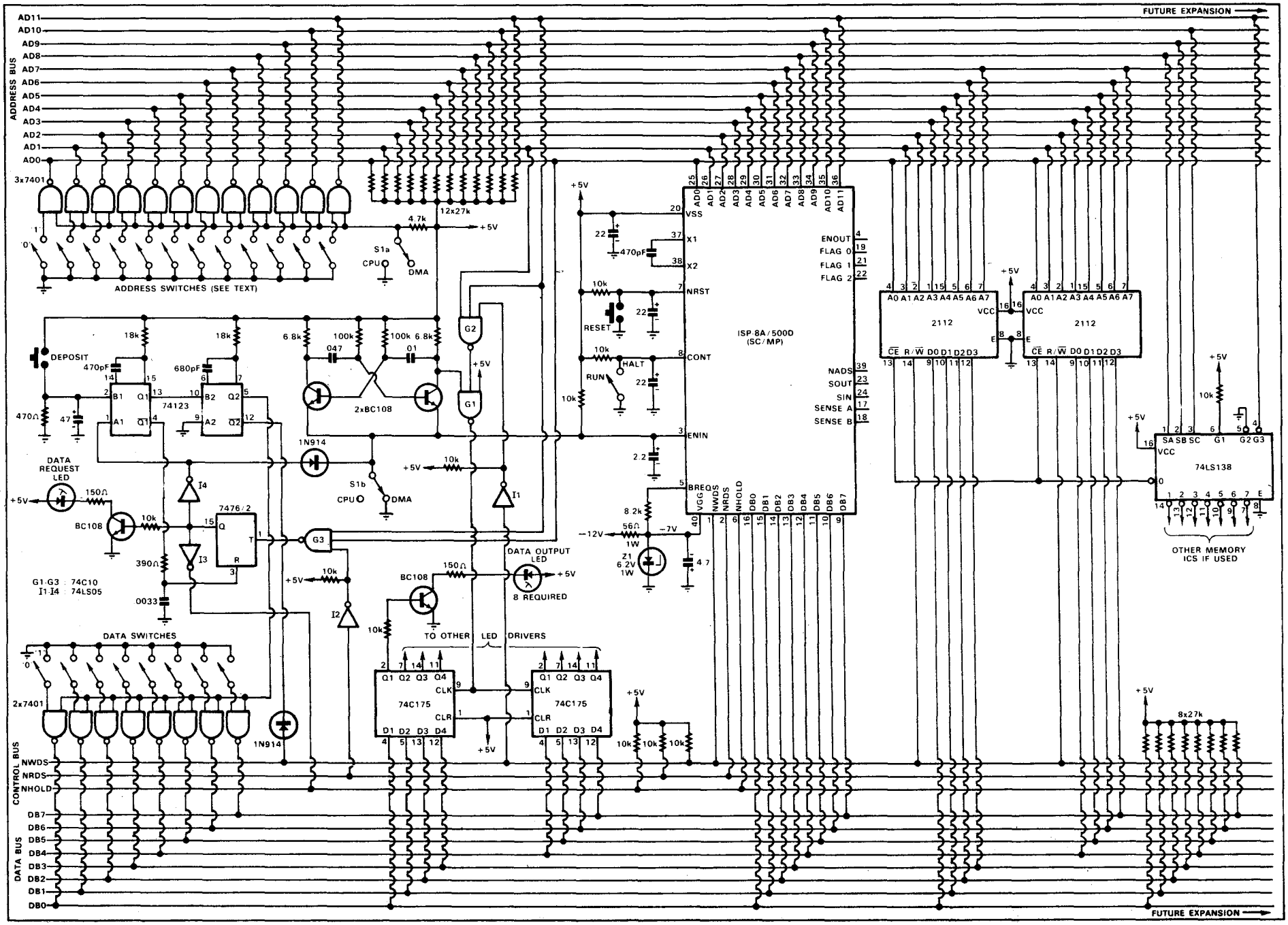
Excluding the power supply, the computer may be conveniently divided into three basic units. These are the central processing unit (CPU), the memory, and the front panel input/output circuitry. These three units communicate with one another via three system 'bus' lines: an address bus, a data bus, and a control bus.

The address bus, comprising twelve actual lines, is used by both the CPU and the front panel I/O circuit to specify which location in memory information is to be sent or received. It is also used by the CPU during program execution, to select a particular input or output device for communication with the outside world.

The data bus, of 8 lines, enables the passage of information between any two of the three units in either direction. The unit that does not participate in a given



At left is the author's prototype of his Mini Scamp, with the full circuit shown on the page opposite.


ADDRESS BUS
AD11
AD10
AD9
AD8
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
FUTURE EXPANSION
3x7401
ADDRESS SWITCHES (SEE TEXT)
12x27k
4.7k
+5V
S1a
CPU
DMA
ISP-8A/500D
(SC/MP)
VSS
X1
X2
NRST
CONT
ENIN
SIN
SOUT
SENSE A
SENSE B
FLAG 0
FLAG 1
FLAG 2
ENOUT
NADS
BREQ
VGG
NWDS
NRDS
NHOLD
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
22
470pF
10k
RESET
HALT
RUN
2.2
8.2k
56Ω
1W
-12V
-7V
Z1
6.2V
1W
4.7
2112
VCC
74LS138
SA SB SC G1 G2 G3
OTHER MEMORY ICS IF USED
DEPOSIT
470pF
680pF
74123
18k
6.8k
100k
047
01
2xBC108
G1
G2
G3
1N914
S1b
DATA REQUEST LED
150Ω
BC108
7476/2
I1
I2
I3
I4
390Ω
0033
G1-G3 : 74C10
I1-I4 : 74LS05
DATA OUTPUT LED
8 REQUIRED
TO OTHER LED DRIVERS
74C175
CLK
CLR
DATA SWITCHES
2x7401
CONTROL BUS
DATA BUS
8x27k
FUTURE EXPANSION

data transfer is disabled (i.e. put in a high impedance state) so that it does not affect the transfer.

The control bus, of only 3 lines, is used to specify whether information is to be read from the memory, or is to be written into the memory, and to place the CPU in a 'hold' condition while it waits for information to be given it via the front panel or other slow peripheral device.

Twelve address lines enable a total of 4096 words of memory and/or peripheral devices to be addressed independently by the CPU. The concept of the bus system described here makes possible the easy expansion of the computer up to this limit, if so desired, by the addition of more memory and more I/O devices. The SC/MP CPU is actually capable of directly addressing up to 65k of memory and/or peripherals. Although this may be readily accomplished with a latch and some buffer IC's it will not be discussed here further.

Of the three sections making up the system the CPU is the heart, or rather the brain, of the system. It comprises the SC/MP integrated circuit microprocessor chip, which requires two voltages for correct operation, +5V and −7V. The −7V (actually −6.2V) is provided from a nominal −12V line by means of a series dropping resistor (56 ohms, 1W) and a zener diode regulator. The other resistors in the circuit are pull-up resistors, to ensure that the appropriate pins on the SC/MP have the correct potential for normal operation.

Some of these potentials can be modified by switches on the front panel. For instance, the DMA/CPU switch can disable the CPU by placing zero potential on the ENIN terminal of SC/MP. This is necessary when data is being entered into memory via a direct memory access (DMA) from other front panel switches, as described later. The RUN/HALT switch controls the potential of the CONTinue terminal, and enables suspension of program execution at any time. The RESET pushbutton must be pressed before initial execution of each program. This ensures that all internal registers of the SC/MP are set to zero, and that the first instruction fetched from the memory will be from location one.

The capacitors on each of these switched lines are crude debounce devices, but have been found to be quite adequate. A quick or snap action when using the switches will always help in this respect.

The 470pF capacitor connected between the X1 and X2 pins determines the speed at which the processor will run. Unlike many other microprocessors, the SC/MP has all the required timing generation circuitry built in, with the exception of this one external component. The value of capacitance shown here will run the SC/MP at near its maximum speed with a 'microcycle' time of 2us. Typical program instructions in the SC/MP take from 5 to 22 microcycles to execute.

The memory section of the circuit uses two low-cost 2112 static MOS memory chips which together provide 256 words of memory, each word of 8 bits in length. These words occupy address locations starting at 0 and extended to 255 (decimal) inclusive (0-FF hexadecimal). The eight address pins on the devices are fed from the eight least significant address lines (AD0-AD7 inclusive).

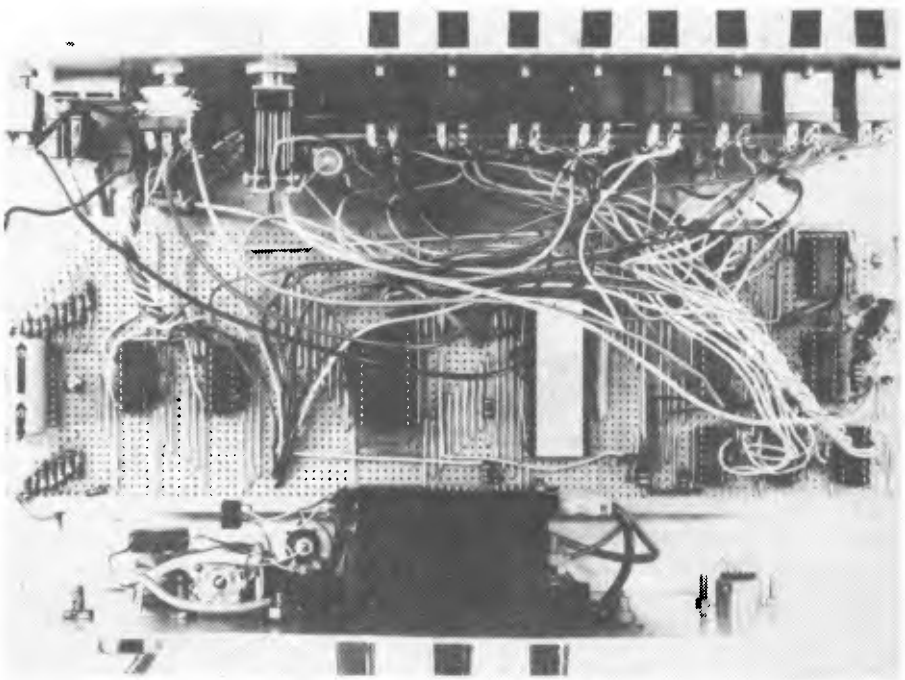
To ensure that the devices only occupy address locations 0-255, the remaining lines of the address bus are fed to a 74LS138 one-of-eight decoder. The "0" output of the decoder is then fed to the chip select (CS) inputs of the memory chips, so that the latter are only enabled or "selected" when the four address lines AD8 through AD11 are in the zero state.

Note that the 74LS138 is basically a 3-bit decoder, and has only three nominal code inputs. The most significant address line AD11 is therefore fed to one of the decoder's own chip select inputs, to achieve the desired result.

Note also that the remaining outputs available on the 74LS138 (1-7 inclusive) may be used to provide selection signals for additional memory devices. The memory of the system can thus be expanded very simply, merely by adding further pairs of 2112 devices.

The front panel I/O section actually has the greatest circuit complexity of the three parts of the system—neglecting, of course, the tremendous internal complexity of the CPU and memory LSI chips. It has two fundamentally different modes of operation. When the DMA/CPU switch is in the DMA position, then the address switches have control of the address bus. The contents of the memory address indicated by these switches will be displayed by the LED's (L0 to L7) on the front panel.

If it is desired to change the contents of any particular memory location, the address of that location is set up on the address switches, and the data to be inserted is set up on the data switches (DS0 to DS7). If the DEPOSIT pushbutton is then depressed and released, the LED's will confirm that the data has indeed been stored in memory at that location. In this way a program may be loaded into memory. This is described in more detail a little later, using a sample program.



The inside of the author's prototype, which was built up using Veroboard. To help readers we are producing a PCB pattern – see box at lower right.

As it is more convenient to represent both data and addresses in hexadecimal rather than binary notation, it will be found convenient to group or delineate these switches into fours. PVC marking tape was used on the front panel of the prototype computer as can be seen in the accompanying photograph.

In the second mode of operation, the DMA/CPU switch is set to the CPU position. In this mode, the address switches are disabled, and have no control over the address bus. The RUN/HALT may then be set to RUN and the CPU will begin to execute whatever program instructions are in memory at this time. Also in this mode, the data switches function as an input device at the hexadecimal address 0801 (hex). Thus, under program control, data can be read into the CPU from the data switches. The

instruction to do this has the form

LD SWITCHES

where SWITCHES has a hex value of 0801. The LD instruction loads whatever data is found at the address of the operand (in this case address 0801) into the accumulator register of the CPU.

To indicate to the external world that it requires data (i.e., that the above instruction has been executed), the CPU, via signals on address lines AD0 and AD11, and on the read data strobe control line (NRDS) is used to turn on a data request LED (DRQ). This is done via G3 which detects a coincidence of the above three signals and toggles the flip-flop which turns on the DRQ LED and also pulls the NHOLD control line to zero. This will cause the CPU to remain in a 'wait' condition until the line returns to a 'one' state. This will occur when the deposit button is pushed, triggering the monos, which after a small delay from the 390-ohm and 0.0033uF RC network, reset the flip-flop.

In this second mode the LED's act as an output device with an address 0802 (hex), which is selected by G2 and activated by an instruction of the form

ST LEDS

where LEDS represents the hex address 0802.

The astable multivibrator comprised of the two BC108's is disabled in the CPU mode, but in the DMA mode provides a continuous string of latching pulses to the 74C175's so that the LED's will always display the contents of the address as indicated by the address switches.

The two diodes in the circuit are used as cheap 'OR' gates for simplification. They could be replaced by another IC if desired, but they have proved quite adequate, and help to keep the cost and total package count down. For similar reasons the 7401 has been used as a "poor-man's tri-state buffer", instead of the more expensive buffer IC's manufactured especially for tri-state applications. There is no reason, however, why these latter chips, such as the 81LS97, should not be used if available.

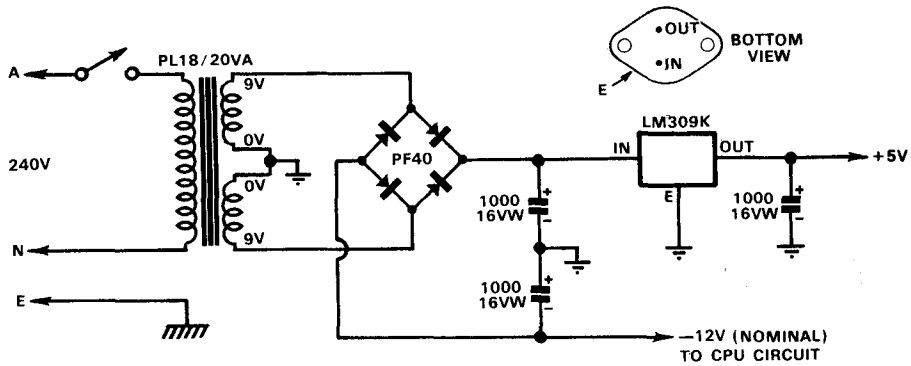


Fig. 2: A simple power supply circuit for Mini Scamp. Any other supply capable of delivering 5V at 500mA and −12V at 150mA could be used instead.

If the system is to be run with only the minimum amount of memory, (i.e., 256 words) then only 8 address switches are required (2^8 = 256). The remaining switch lines (AS8 to AS11) may simply be left floating. This is equivalent to placing a zero on these lines. Thus, depending upon the amount of memory you have available, anywhere from 8 to 12 address switches will be required on the front panel.

It should be noted however, that no more than 2K of memory can be accommodated on this system without modification. This is because addresses above hexadecimal 0800 (or decimal 2048) are used to reference the data switches and LED's.

Those of you who have closely followed the circuit will have realised that in fact the addresses 0801 and 0802 (hex) are not unique in their ability to reference the switches and LED's respectively. All twelve address lines instead of only two, as used at present, would be necessary to uniquely specify a single device. Thus all addresses which have a one in bit positions 0 and 11 will refer to the data switches. These all lie at locations greater than 0800 (e.g. 0803, 0805, 08A9, etc.) and so will not conflict with memory addresses less than this value.

Various techniques and methods may be used in the construction of the microcomputer. Veroboard was used to build the prototype as shown in the photograph, and probably provides the lowest cost way to go. Layout and component placement is not critical. A single length of Veroboard could be used, or the three main sections could each be constructed on smaller separate boards. This latter approach allows one to more easily interchange units if desired. (e.g. to try an 8080 CPU, or to substitute a larger memory unit).

Printed circuit boards for each section make for simpler construction and greater flexibility, and several members of our microcomputer club will probably employ this approach. However, it should be borne in mind that the cost of PCB's and their respective sockets will

Mini Scamp: a PCB is coming

Dr. Kennewell's "Mini Scamp" microcomputer design seems to us to be just what many of our readers have been waiting for: a really simple way of becoming familiar with microprocessors and their operation. Because of its low cost, its ease of expansion, and the fact that it needs no expensive terminal, we believe it could become an extremely popular project and a worthy successor to our own EDUC-8 design. To help ensure this well-deserved popularity, we are producing a low-cost PCB pattern for the project. All going well we hope to publish details next month.

Sydney micro club

The Microcomputer Enthusiast Group or "MEGS" meets on the first and third Mondays of the month at the WIA centre, 14 Atchison Street North Sydney. Although formed only a few months ago, the group is already running member training courses on both hardware and software, and publishing a monthly newsletter. It has also set up committees to look into such matters as hardware compatibility.

The group has received welcome support from local suppliers, At the time of writing this, two suppliers had donated microcomputer evaluation kits, while other firms have sent applications engineers to offer literature and advice.

At the invitation of MEGs, EA editor Jim Rowe went to their meeting on April 18th. Also present that evening were NS engineers Ed Schoell and Chris Mason, who demonstrated some of their firm's PACE and SC/MP systems. There was a good attendance at the meeting, with some 60 or 70 keen enthusiasts present.

EA editor Jim Rowe took the Mini Scamp computer and EA video terminal to the meeting, and these seemed to produce quite a bit of interest from members.

Membership inquiries to: MEGS, c/o Post Office, St. Leonards 2065.

```
                *BINARY COUNT AND DISPLAY
0000 08             NOP
0001 C408           LDI   8
0003 35             XPAH  1
0004 C400           LDI   0
0006 31             XPAL  1
0007 C902   LOOP    ST    2(1)
0009 8FFF           DLY   255
000B B803           ILD   COUNT
000D 90F8           JMP   LOOP
000F   00   COUNT   .BYTE 0
0010
```

```
                *MOVING LIGHTS WITH INPUT
0000 08             NOP
0001 C408           LDI   8
0003 35             XPAH  1
0004 C400           LDI   0
0006 31             XPAL  1
0007 C101   LOAD    LD    1(1)
0009 C810           ST    BITS
000B C00E   LOOP    LD    BITS
000D C902           ST    2(1)
000F 1E             RR
0010 C809           ST    BITS
0012 8FFF           DLY   255
0014 B806           DLD   COUNT
0016 9CF3           JNZ   LOOP
0018 90ED           JMP   LOAD
001A   00   BITS    .BYTE 0
001B   00   COUNT   .BYTE 0
```

Here are two sample programs to help you get going with Mini Scamp. In both cases the hexadecimal numbers in the first column are memory addresses, and those in the next column are the actual code. Each pair of hex digits is an 8-bit byte.

greatly increase the cost of the overall unit, and may not be justified, particularly if further expansion is not desired.

The LED's and their transistor drivers were soldered onto a long narrow strip of Veroboard and then the LED's were glued through holes in the front panel.

The power supply requirements are quite small, and any supply giving +5V at say 0.5A and about -12V at 150mA should prove adequate. Fig. 2 gives a typical circuit for those wishing to build the power supply using new components.

It will be found that the cost of the switches can be a considerable fraction of the total computer cost. Those used for the prototype were lever switches (DPDT) from Tandy Electronics. Similar switches at a much lower cost from Electronic Disposals in Little Lonsdale Street in Melbourne have also been tried. Although satisfactory to date, only time will allow us to determine how many repeated switchings may be made before the contact resistance becomes too large for correct operation. In this regard, it is most desirable to wire both poles of the above type of switches in parallel.

Although this microcomputer was conceived mainly as an educational instrument through which an understanding of the engineering and programming concepts involved could be learnt, there is no reason why it could not be put to work in the role of a simple controller. Detection of off/on states of various devices, and the activation of relays, etc., is most easily accomplished using the sense inputs and flag outputs available on the SC/MP chip. Anyone wishing to make good use of the computer should obtain a copy of the SC/MP Technical Description from NS Electronics Pty. Ltd., in Bayswater, Victoria, or from their distributors. This manual describes all of the program instructions available on the SC/MP, and details what each of these actually does.

On the programming side, you might like to try your hand at writing a multiplication routine, a BCD to binary conversion program, the converse, i.e. binary to BCD, or even a simple program to demonstrate the function of the logical operations, AND, OR and EXCLUSIVE OR.

Although it uses a different instruction set than does the SC/MP, the advice on programming contained in the EDUC-8 handbook provides valuable information for those with little prior knowledge in this field. I have also found the hexadecimal conversion table printed in the E.A. Yearbook (1976/77) to be of great assistance when manually assembling small programs.

In order to get you started along the road in programming your microcomputer, I will describe two short demonstration programs.

The first program simply counts in binary, displaying each number on the LED's with a fixed delay between numbers. The delay is necessary to slow the computer down sufficiently for you to observe what is happening. The program listing is shown in Fig. 3.

Ignoring the first two columns for the moment, what we have is a list of instructions to the computer in 'Assembly' language. The first instruction (NOP) does nothing, and is ignored by the CPU, as the first instruction actually executed is at address one. The next four instructions load the hex address 0800 into pointer register 1. The following instruction (ST 2(1)) outputs whatever number is presently in the accumulator to the LED's. Note that the operand 2(1) means the address stored in pointer register 1 plus 2 (i.e., 0802) which is, of course, the address of the LED's.

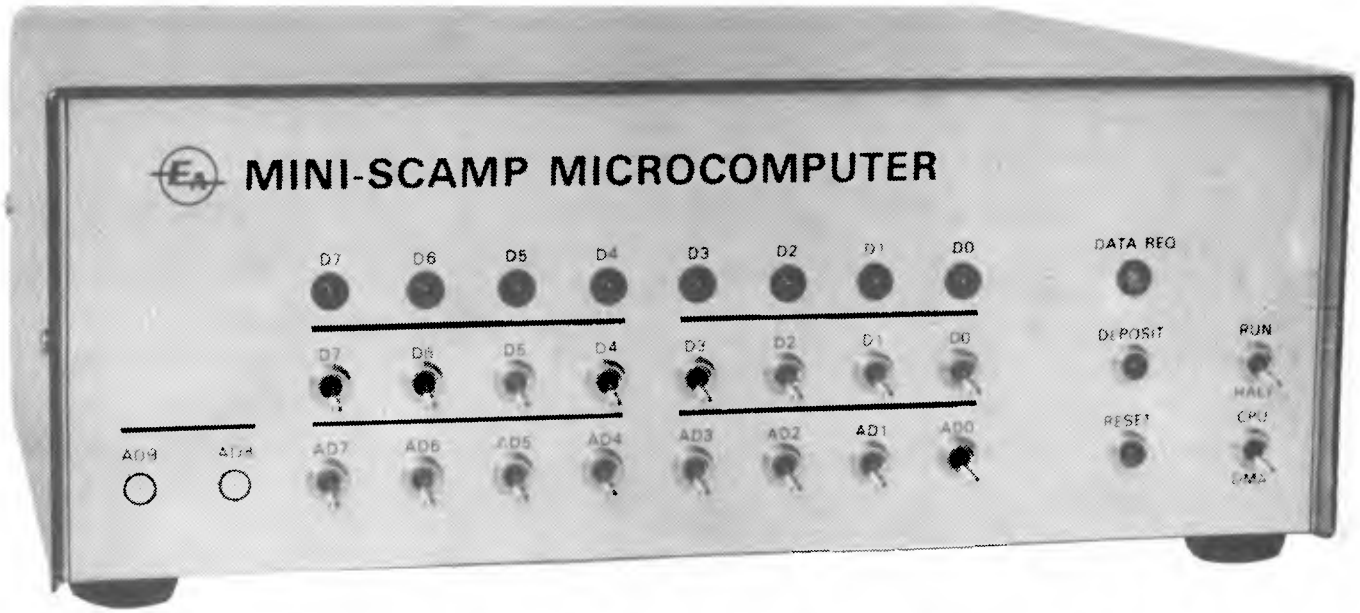
The next instruction (DLY 255) creates the delay, while the ILD COUNT instruction adds one to the location called COUNT (which has address 000F) and then loads this number into the accumulator, ready for display on the LED's when the CPU jumps back (via the JMP LOOP instruction) to the ST 2(1) instruction.

The information in the second column is the translation of the assembly language instructions detailed above into machine language form. These hexadecimal numbers may be loaded into the memory at their respective addresses shown alongside in column 1.

First, set the RUN/HALT switch to HALT, and the DMA/CPU switch to DMA. Then set all the address switches to zero, and set the hexadecimal number 08 on the data switches. Now press DEPOSIT, and the LED's should also display 08. Continue by setting the address switches to 01 and the data switches to C4. Depress DEPOSIT again. As the LDI 8 instruction is a double-byte instruction, the number 08 must now be set into address 02 followed by 35 into 03, and so on, using the above procedure. When all locations up to and including 0F have been loaded, the program is ready to be executed or run.

Set the DMA/CPU switch to CPU, depress and release the RESET button, and then set the RUN/HALT switch to RUN, and the LED's should then be counting. If not, return the appropriate switches to HALT and DMA, in that order, and check the contents of each address by successively incrementing the address switches from 00 to 0F hex.

The second program shown in Fig. 4 demonstrates both the data input facility of the computer and also the rotate instruction (RR). When run, the program will request (via DRQ) any number from the data switches. For example, when DRQ comes on, set hex 80 on the switches, then press DEPOSIT. The single light on the left will then be successively moved to the right and finally 'rotated' back to its initial position. After 256 rotations, the program will then request a new bit pattern to rotate.



Look what happened to the Mini Scamp!

With some help from component suppliers, we have been able to turn Dr. Kennewell's Mini Scamp microcomputer design into a complete full-scale construction project. With almost all of the circuitry on a single PC board, it is now not just the lowest-cost complete microcomputer system available, but the easiest to build as well!

by JAMIESON ROWE

Just about everyone interested in microcomputers seems to agree that Dr. John Kennewell's Mini Scamp design has great potential. By starting from scratch with a SC/MP chip, and then designing a simple RAM-orientated system around it, he has produced an ideal microcomputer for the hobbyist and student.

It is fully self-contained, needing no expensive terminal. Programs are fed in via front-panel switches and LEDs, which can also be used to communicate with the machine when it is running—in simple binary code, the actual language used by the machine itself. What better way to learn how computers work!

At the same time, it can be built for around half the cost of any other microprocessor based system, and hundreds of dollars less than broadly comparable earlier designs like our own EDUC-8.

In other words, it is a design which should appeal to a wide variety of people, especially those still looking for a way of becoming familiar with microcomputers easily and at low cost.

While we were preparing Dr. Kennewell's article for last month's issue, the conviction grew that the project deserved to become a very popular one. But we realised that one thing was lacking: a low-cost PC board, to make it really easy to build even for those with little previous experience.

We immediately resolved to design a PCB for the project, to help ensure that it wins the popularity it deserves. And we managed to fit a small "stop press" box in the April article, to let readers know that a PCB was on the way.

Because of the box no doubt quite a few readers have been waiting for the current issue, for the promised PCB design. As you can see, we have in fact gone much further than this, and have turned Mini Scamp into a full-scale project. So that your wait should not have been in vain.

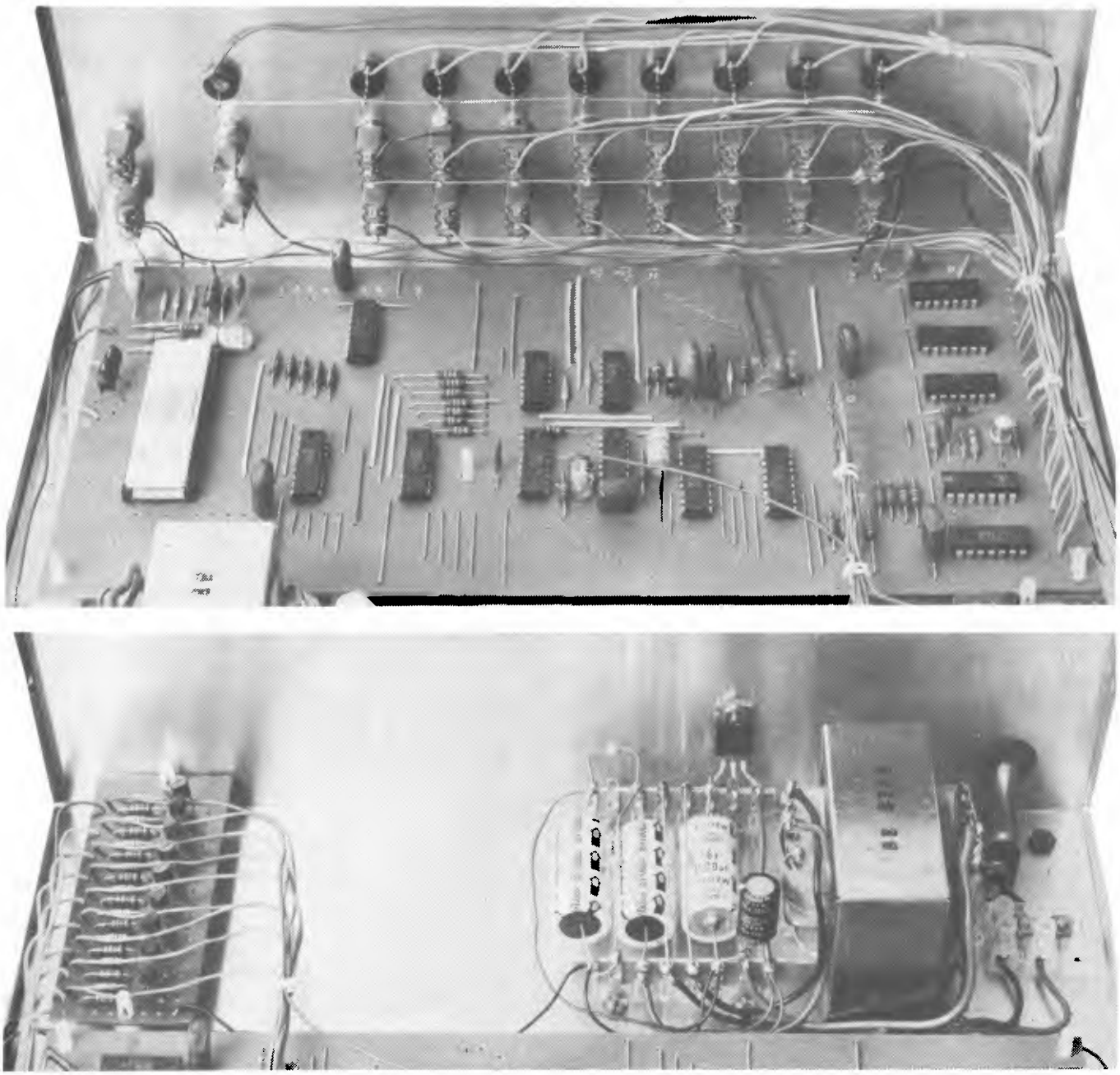
How did this happen? Well, we couldn't publish the PCB design without trying it out first, to make sure it worked. This meant getting together a set of ICs, switches and other components, and at the same time warning suppliers that the project was coming—to ensure that readers would be able to buy the parts. While we were doing this we sought reactions from suppliers also, to see if they became as keen about the project as we were.

They certainly did. In fact, some of the suppliers were so enthusiastic that they offered to make some of the components available to readers at special prices.

For example NS Electronics have offered to make available through their distributors a special deal on the SC/MP chip, all the other ICs, the transistors and LEDs. These will be available for $36.21 including tax, or $31.49 to schools and colleges who can claim a tax exemption, until the end of June.

Similarly C&K Electronics are prepared to supply the complete set of 18 toggle switches and 2 pushbuttons direct to readers for a package-deal price of $14.65 plus 50c postage, or $12.74 plus 50c postage for those who can claim a tax exemption. Their address is PO Box 101, Merrylands, NSW 2160.

When we told the story to well known kits-n-bits entrepreneur Dick Smith, his immediate question was why we weren't



As these pictures show, Mini Scamp now really looks the part, comparing well with machines costing much more. You should be able to build it for around $105 including tax, and thanks to the new PC board it should take you only a few evenings' work!

planning to describe such an excellent design as a full-scale construction project. This would then allow his firm and, others to produce a complete kit...

Needless to say, we decided there and then to do just that. And thanks to quite a bit of help from Dick Smith, NS Electronics, C&K, RCS Radio, Radio Despatch Service and Bespoke Metalwork, we have been able to produce the full project in double-quick time.

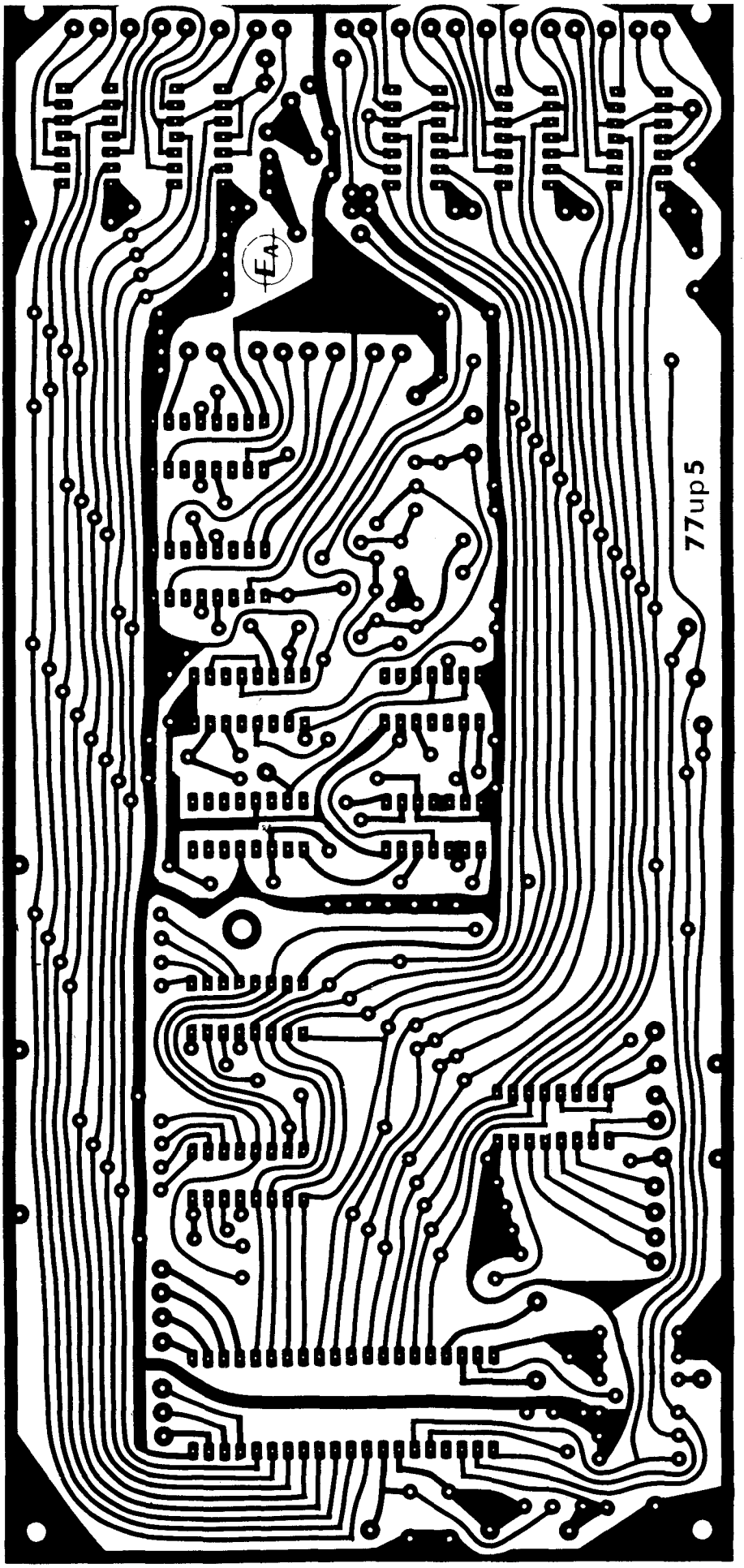
As you can see, it really looks the part, comparing very favourably with designs costing three and four times the price. Yet with most of the circuit on a single PCB measuring 254 x 117mm, you should be able to wire it up very easily in a few evenings.

The basic design has 256 bytes of RAM memory, plenty to let you cut your teeth on basic programming. However we have provided the PCB with data bus, address bus and control signal access, so that additional "outboard" memory may be added very easily. In fact all you will need to expand the memory to 1k bytes is six more 2112 memory chips, a small PCB or piece of perf-board to mount them on, and some hookup wire!

We have even allowed for a couple of additional address switches to be mounted on the front panel, if you want to expand the memory to 1k in this way.

We have also brought out all of the flag and sense pins on the SC/MP chip itself, so that it should be possible to interface Mini Scamp to a terminal later on if you wish. If there is sufficient reader interest we may be able to tell you how to add the "Kitbug" ROM into the system, with its terminal interfacing and debug routines. This should again be a relatively simple matter.

Wiring up the PCB should be quite straightforward as we have prepared an overlay diagram showing the position and orientation of all parts. There is a reasonable number of links, as the PCB is single-sided to keep the cost low, but not so many as one might have expected.



PARTS LIST FOR OUR MINI SCAMP

1 Case, 285 x 235 x 104mm
1 Printed circuit board, 254 x 117mm, coded 77up5
18 SPDT miniature toggle switches, C & K type 7101 or similar
2 Miniature pushbuttons, single pole normally open type, C & K type 8532 or similar
1 stepdown transformer, with multi-tapped 15V secondary at 1A. Type 2155 or similar

SEMICONDUCTORS

1 ISP-8A/500D microprocessor (SC/MP)
2 2112 memory chips (256 x 4)
2 74C175 quad latches
1 74C10 triple 3-input gate
1 74LS138 decoder
1 74LS05 hex inverter
5 7401 hex inverters
1 7476 dual flipflop
1 74123 dual monostable
12 BC108, BC317 or similar transistors
9 5mm diameter LEDs with panel adapters (8 red, 1 green or yellow)
1 LM340T-5, 7805 or similar 5V/1A 3-terminal regulator
4 1N914, 1N4148 or similar diodes
4 50V/1A rectifier diodes
1 6.8V/1W zener diode

RESISTORS

1W rating: 1 x 56 ohm
¼W rating: 9 x 150 ohm, 1 x 390 ohm, 1 x 470 ohm, 1 x 2.7k, 2 x 4.7k, 2 x 6.8k, 1 x 8.2k, 19 x 10k, 2 x 18k, 20 x 27k, 2 x 100k

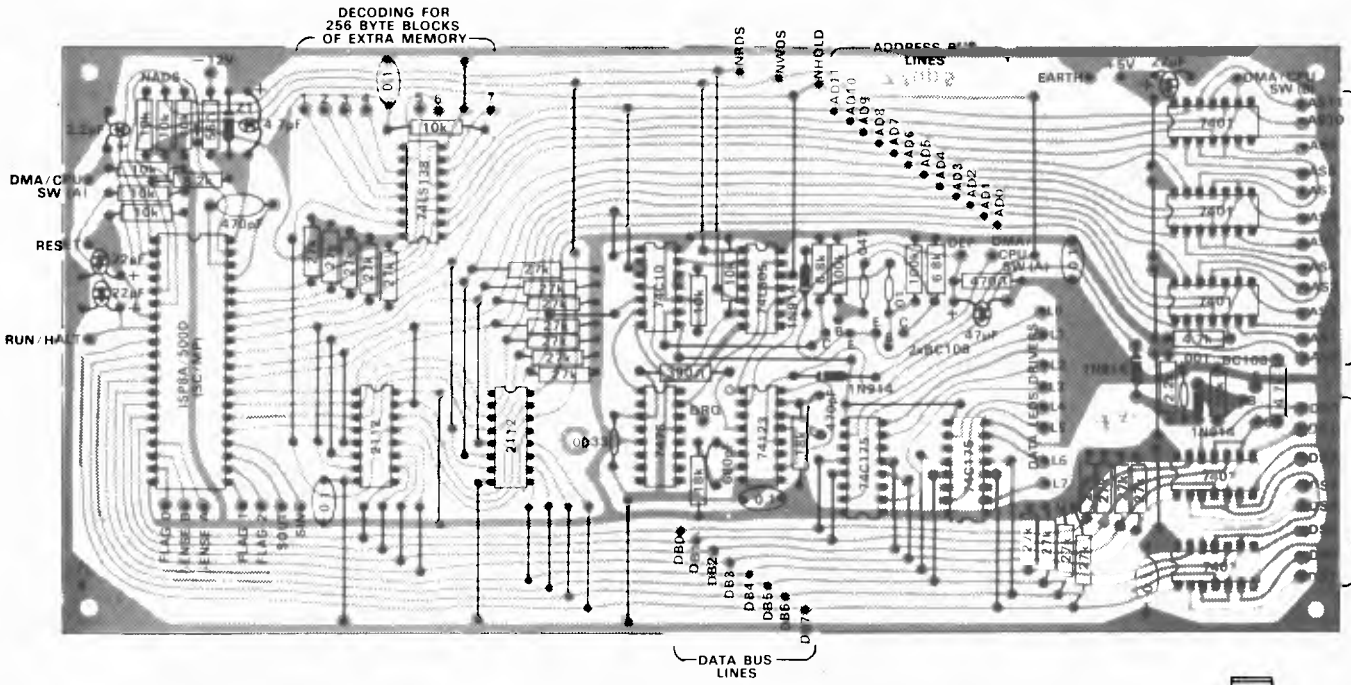
CAPACITORS

LV greencap polycarbonate: 1 x 1000pF, 1 x 3300pF, 1 x .01uF, 1 x .047uF, 5 x 0.1uF
2 470pF polystyrene or NPO ceramic
1 680pF polystyrene or ceramic
1 2.2uF 35VW tantalum
1 4.7uF 35VW tantalum
3 22uF 6VW tantalum
1 47uF 6VW tantalum
1 100uF 16VW electrolytic
3 1000uF 16VW electrolytic

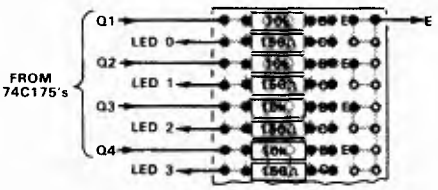
MISCELLANEOUS

Three-wire mains cord and 3-pin plug; grommet and cord clamp; 40-pin DIP socket for SC/MP (PC type); 7 x nylon PCB supports (Richco); 1 x 85 x 40mm piece of utility PCB for LED drivers; 2 x 8-lug miniature tagstrips for power supply wiring; 4 rubber feet for case; connecting wire, solder, nuts, bolts, etc.

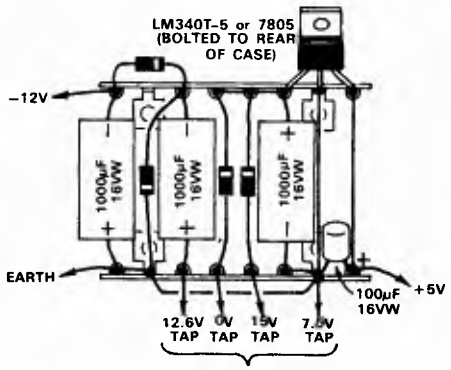
At left is the PCB pattern reproduced actual size. Etched and drilled boards should be available from the usual suppliers by the time you read this.



Wiring up the PCB should be fairly straightforward using the above diagram as a guide. Note that the address and data bus lines are brought out only for future expansion; this applies also to the flag and sense lines. Details of the LED driver and power supply wiring are shown below and at right.



HOW TO WIRE THE LED DRIVER PANEL (9 DRIVERS IN ALL)



POWER SUPPLY WIRING

Incidentally the PCB hole spacing for the SC/MP clock capacitor is 12.5mm, so that readers who wish to substitute a 1MHz quartz crystal for the existing 470pF capacitor can do so easily. This would perhaps be advisable when and if you wish to interface the Mini Scamp to a terminal, to ensure a stable and predictable data rate.

Except for the SC/MP chip itself, all of the ICs are mounted directly on the PCB without sockets. A high-quality 40-pin socket was used for the SC/MP because of its higher unit cost.

We have not included the LED driver transistors and their associated resistors on the main PCB. This would have committed builders to using the binary LED scheme, and we think that some may prefer to use a pair of 7-segment displays with hexadecimal drivers instead. By leaving the drivers off the PCB, you can take your pick.

The original 9-LED scheme has been retained for our version of Mini Scamp, as you can see. This is because we think beginners find a simple binary display easier to follow. It will also be somewhat cheaper than a hex display!

We wired the 9 driver stages on a small piece of utility PCB, of the type having an array of 4 linked-pad groups. By cutting some of the conductors to form pairs, the drivers were very easily wired, as shown in the small diagram. The piece of utility PCB measures only 85 x 40mm.

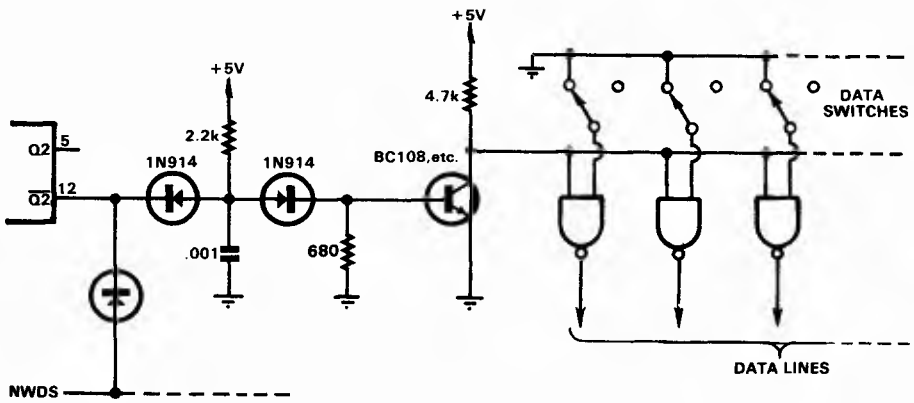
To reduce costs we elected to use one of the imported DSE 2155 transformers. As this provides only 7.5V per side on the secondary, we were unable to use the twin full-wave rectifier circuit suggested by Dr. Kennewell last month. The positive rectifier is unchanged, but we have substituted a half-wave doubler for the negative supply. This runs from the "12.6" tap (i.e., 5V with respect to the earthed centre-tap), to produce the desired –12V.

The only other change to the power supply is that we found it necessary to add a 100uF/16VW electro across the output of the 5V regulator. The wiring of the final supply is shown in a small diagram, so you can copy it if you wish.

One further point: you will find that the PCB incorporates a change to the memory deposit circuit. In particular, we have added an RC delay circuit and a buffer transistor to the drive line for the data switch gates, as shown in the diagram at left. Drive is now taken from the Q2-bar output of the 74123 (pin 12), instead of from the Q2 output.

We found this modification necessary to ensure fully reliable depositing with the 2112 memory devices, which have a critical requirement in respect to data hold time.

Well, there you have it. We think you'll agree that Mini Scamp has become quite an exciting project—and an excellent way to learn about microcomputers.



DEPOSIT TIMING MODIFICATION

Making Mini Scamp bigger and better

Here are the promised details showing you how to expand the basic Mini Scamp microcomputer design into a more powerful one. You can interface it to a teleprinter or a video terminal very easily, and you can also expand its memory to either 1,024 bytes of RAM or 1,280 bytes of mixed RAM and ROM or PROM—all at surprisingly low cost!

by JAMIESON ROWE

In its original form, Dr. Kennewell's Mini Scamp microcomputer is excellent for teaching the fundamentals of microcomputer operation. However once you have learned the fundamentals, it can be a bit frustrating to have to feed your programs in each time via the front panel switches and LEDs. Before long, you will want to add interfacing so that you and your Mini Scamp can talk to each other more conveniently, using a teleprinter or video terminal.

As it happens, you can provide your Mini Scamp with standard 20mA-loop asynchronous serial interfacing very easily, and at low cost. All that is required are two low cost transistors and a handful of other parts, connected to the "Flag-0" and "Sense-B" pins of the SC/MP chip itself (pins 19 and 18 respectively).

The circuit details are shown in Fig. 1. As you can see, the output interfacing from the SC/MP flag pin is simply a BC558 or similar PNP transistor arranged as a switch controlling a 20mA current derived from the +5V line. When the SC/MP flag is at the logic low level, the transistor is turned on and 20mA of current flows out into the teleprinter or terminal. This is the "mark" or idle condition.

If the flag is taken to logic high, the transistor is turned off. The output current to the terminal is thus interrupted, producing the "space" condition. The SC/MP is thus provided with a means of switching the terminal current between the standard mark and space levels, via the level at its flag 0 output.

The input side of the interfacing is equally simple. The keyboard output of the terminal is used to control the conduction of another transistor switch, this time a BC548 or similar NPN device. This is connected in turn between the SC/MP sense-B input and ground, so that it controls the logic level at this input (there is an internal pull-up resistor of around 7.5k, on the SC/MP chip).

The RC circuitry at the input of the transistor is used to suppress contact bounce and any hash which may be picked up by the terminal cable.

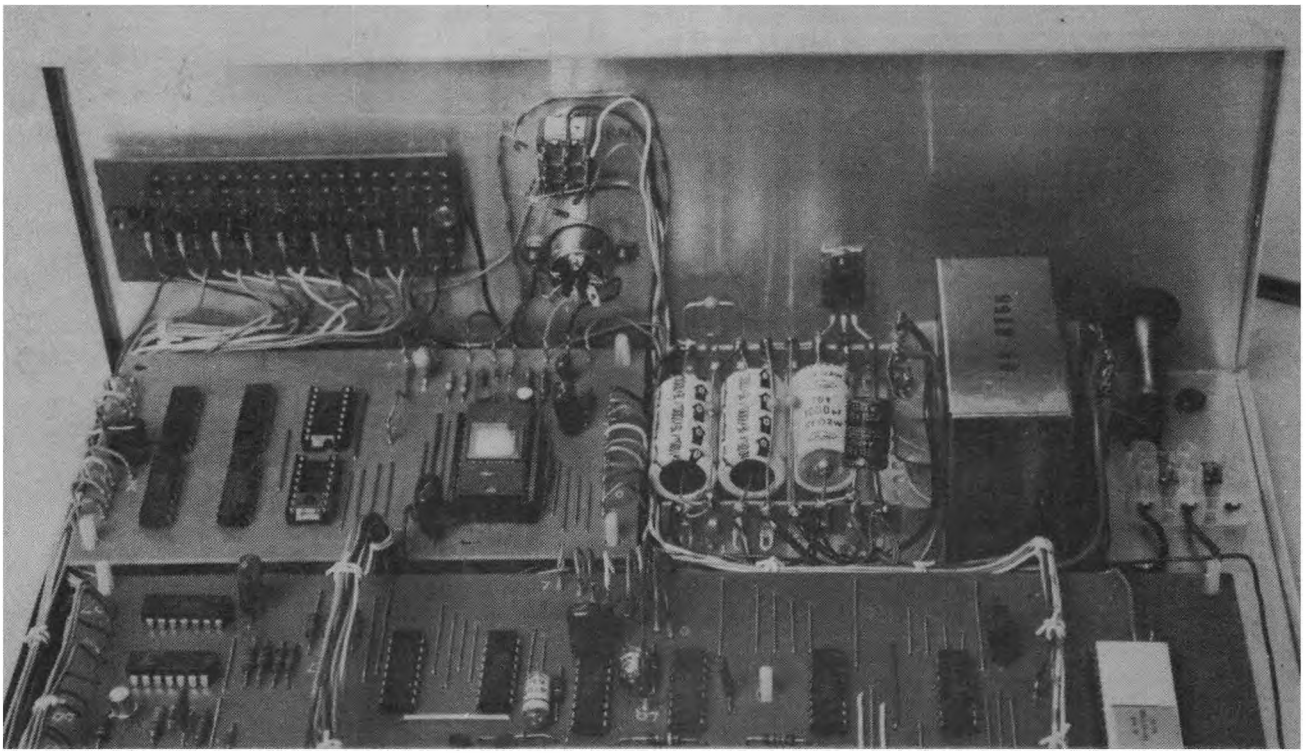
When the keyboard output of the terminal is in the low-impedance "mark" state, it holds the base of the transistor at ground potential. The transistor is thus cut off, and the pullup resistor inside the SC/MP chip pulls the sense-B input up to the logic high level.

If the keyboard output switches to the high impedance "space" state, a forward bias is applied to the transistor base via the divider formed by the 1k, 3.3k and 1k resistors. The transistor thus conducts, pulling the SC/MP sense-B input down to the logic low level.

Hence this simple interface allows a standard teleprinter keyboard or video terminal output to communicate with the SC/MP chip, by controlling the logic level at its sense-B input.

As you can see, the interfacing circuits of Fig. 1 are very basic. They merely provide electrical matching between the SC/MP flag-0 and sense-B pins and a standard 20mA current-loop terminal. There is no special hardware for handling information in asynchronous serial form, or for converting between this form and parallel format. This is all left to the SC/MP itself to take care of, via suitable software "driver" routines.

As well as being very economical, this



The pictures above and on the facing page show the prototype Mini Scamp in expanded form. It now sports a serial terminal interface, a total of 768 bytes of RAM, and a 512-byte ROM with resident monitor.

approach is also a very flexible one. It means that you can change the serial data format very easily, simply by changing the software driver routines. All you need to do to interface Mini Scamp to a terminal with a particular code and data rate is write the appropriate driver routines!

What this means, for example, is that you aren't just limited to using a relatively expensive ASCII-code teleprinter or video terminal. You can use a surplus Baudot-code teleprinter instead, merely by writing a pair of driver routines to suit its 5-bit data format and 50 (or 45) baud data rate.

So that you won't be completely on your own, we are reproducing here listings for a pair of driver routines for talking to a standard 110-baud ASCII teleprinter or video terminal. These should at least give you an idea of what is involved, so that you can write similar routines for other types of terminal.

Both routines are reproduced here by courtesy of National Semiconductor, as they are basically those which come in the "KITBUG" monitor ROM. We have modified them slightly to adapt them for general use in RAM, and deleted the original ROM addresses as these are not relevant.

GECO is the input subroutine, which is called to fetch a character from the terminal and return with it in the SC/MP accumulator. It also echoes the character automatically to the teleprinter or video terminal display screen.

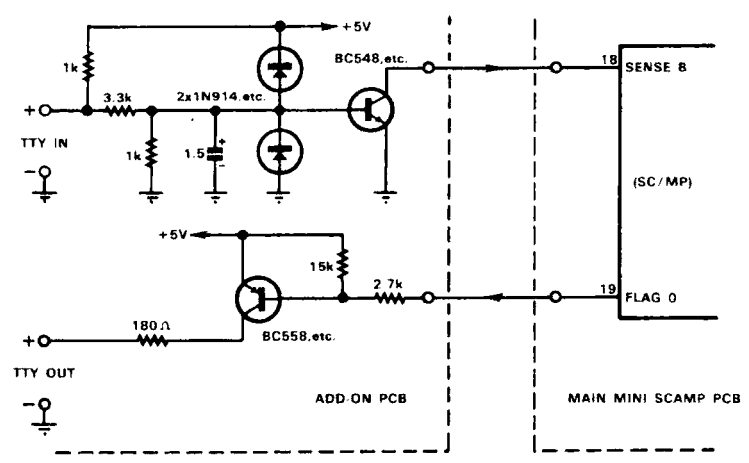


FIG. 1 TELEPRINTER/VIDEO TERMINAL INTERFACING

GECO expects pointer register P2 to have been set up as a pointer to a stack in RAM—i.e., it expects P2 to contain the address of the uppermost of a small stack of "scratchpad" memory locations.

You call GECO using an XPPC P3 instruction (hex code 3F), having first set pointer register P3 to the address of the memory byte immediately before the start of GECO.

As you can see GECO returns to your program by performing a similar XPPC P3 instruction. Because this leaves P3 pointing to its own exit, the "JMP GECO" instruction at the end allows you to call the routine again simply by using another XPPC P3 instruction. (In effect, the start of the subroutine for repeated calling is at the end—this is one of the little tricks with SC/MP.)

The second routine PUTC is to take a character in the accumulator, and send it to the terminal. Like GECO, PUTC is called using an XPPC P3 instruction, but of course P3 must in this case have been set up to the address of the byte immediately before either the first or last instruction of PUTC, rather than those for GECO.

PUTC also expects P2 to be set up as a stack pointer, like GECO. In fact if you use the two routines with a program, it is a good idea to use P2 as the stack pointer for your own program, to avoid hassles. It is often convenient to allocate the very top of the RAM as the stack, so that P2 is set up by loading it with the

```
      ; GECO IS USED FOR KEYBOARD INPUT SO IT ECHOS THE
      ; CHARACTER BUT DOES NOT ENABLE THE READER RELAY.
      ;
C408  GECO:   LDI   8          ; SET COUNT = 8
CAFF          ST    -1(P2)
06    $2:     CSA              ; WAIT FOR START BIT
D420          ANI   020
9CFB          JNZ   $2         ; NOT FOUND
C457          LDI   87         ; DELAY 1/2 BIT TIME
8F04          DLY   4
06            CSA              ; IS START BIT STILL THERE?
D420          ANI   020
9CF2          JNZ   $2         ; NO
06            CSA              ; SEND START BIT (NOTE THAT
DC01          ORI   1          ;   OUTPUT IS INVERTED)
07            CAS
C47E  $LOOP:  LDI   126        ; DELAY 1 BIT TIME
8F08          DLY   8
06            CSA              ; GET BIT (SENSEB)
D420          ANI   020
9802          JZ    $3
C401          LDI   1
CAFE  $3:     ST    -2(P2)     ; SAVE BIT VALUE (0 OR 1)
1F            RRL              ; ROTATE INTO LINK
01            XAE
1D            SRL              ; SHIFT INTO CHARACTER
01            XAE              ; RETURN CHAR TO E
06            CSA              ; ECHO BIT TO OUTPUT
DC01          ORI   1
E2FE          XOR   -2(P2)
07            CAS
BAFF          DLD   -1(P2)     ; DECREMENT BIT COUNT
9CE5          JNZ   $LOOP      ; LOOP UNTIL 0
06            CSA              ; SET STOP BIT
D4FE          ANI   0FE
07            CAS
8F08          DLY   8
40            LDE              ; AC HAS INPUT CHARACTER
D47F          ANI   07F
01            XAE
40            LDE
3F            XPPC  P3         ; RETURN
90C1          JMP   GECO
```

```
      ; 'PUTC'
      ; PUT CHARACTER  IN AC TO TTY.
      ; NOTE: TTY LOGIC LEVELS ARE INVERTED FOR OUTPUT
      ;
01    PUTC:   XAE
C4FF          LDI   255
8F17          DLY   23
06            CSA              ; SET OUTPUT BIT TO LOGIC 0
DC01          ORI   1          ;   FOR START BIT. (NOTE INVERSION)
07            CAS
C409          LDI   9          ; INITIALIZE BIT COUNT
CAFF          ST    -1(P2)
C48A  $1:     LDI   138        ; DELAY 1 BIT TIME
8F08          DLY   8
BAFF          DLD   -1(P2)     ; DECREMENT BIT COUNT.
9810          JZ    $EXIT
40            LDE              ; PREPARE NEXT BIT
D401          ANI   1
CAFE          ST    -2(P2)
01            XAE              ; SHIF DATA RIGHT 1 BIT
1C            SR
01            XAE
06            CSA              ; SET UP OUTPUT BIT
DC01          ORI   1
E2FE          XOR   -2(P2)
07            CAS              ; PUT BIT TO TTY
90E8          JMP   $1
06    $EXIT:  CSA              ; SET STOP BIT
D4FE          ANI   0FE
07            CAS
3F            XPPC  P3         ; RETURN
90D4          JMP   PUTC
```

Here are two utility driver routines, designed to service a 110-baud ASCII teleprinter or video terminal. Both assume that the SC/MP clock oscillator is running at 1MHz, so that if you aren't using a crystal you may need to "tweak" the clock capacitor.

address of the top of your RAM area.

And talking about RAMs leads us on to the next area of interest when it comes to expanding Mini Scamp.

Although the 256 bytes of memory provided in the basic Mini Scamp are enough to let you work on quite respectable machine-language programs, the odds are that it won't take long before you'll be wanting to expand the memory to run larger programs. This seems to happen with almost everybody.

As we noted last month, it is quite easy to expand Mini Scamp's memory up to quite a respectable level. In fact if you want to simply provide additional RAM, it is merely a matter of connecting additional pairs of 2112 memory chips up to the address and data buslines, the write strobe line and the appropriate outputs of the 74LS138 address decoder.

You can add up to three additional pairs of RAM chips in this simple way, without exceeding the loading capability of the SC/MP chip. This will give you a total of 1,024 bytes of RAM, or "1k", which should be more than adequate for any machine language program you're likely to write for quite a while.

The basic connections required for this sort of simple RAM expansion are shown in Fig. 2. As you can see, the existing RAMs on the main Mini Scamp PCB remain as the lowest 256 bytes, gated by the 0 output of the decoder as before. The additional pairs are gated by the next three decoder outputs, so that the four pairs provide a continuous range of addresses from 000 to 3FF hexadecimal.

Note that you can't expand to beyond 1k of RAM using this approach, because any further RAM pairs would exceed the SC/MP loading capacity. More about this later . . .

Naturally enough some readers may like to provide their Mini Scamp with the ability to run a program in ROM or PROM memory. For example, you may have an MM5214 ROM with the "Kitbug" monitor program, as supplied with the National Semiconductor SC/MP kit.

Although Kitbug is a fairly basic little monitor program, it does include the terminal driver subroutines—so that your programs can simply call them as required. It also has another subroutine

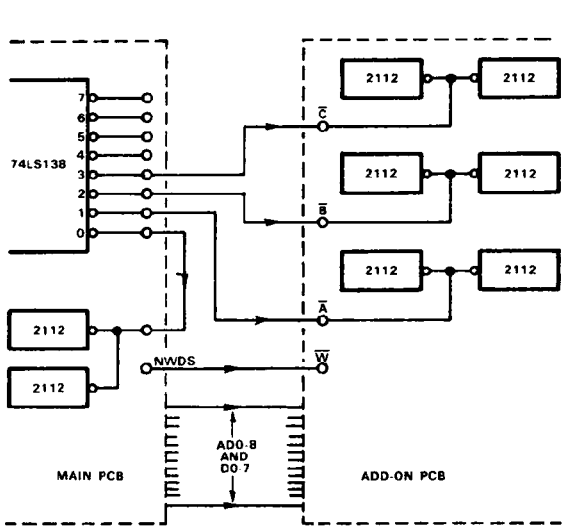


FIG. 2 DECODING FOR RAM-ONLY SYSTEM — TOTAL OF 1k (000-3FF)

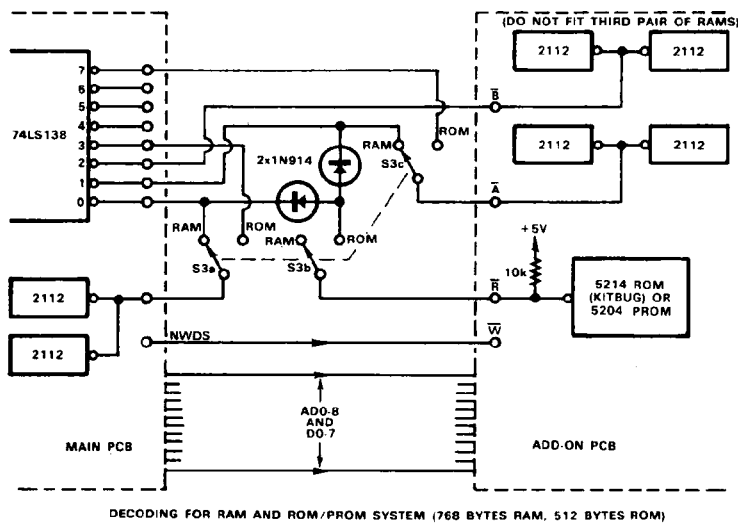
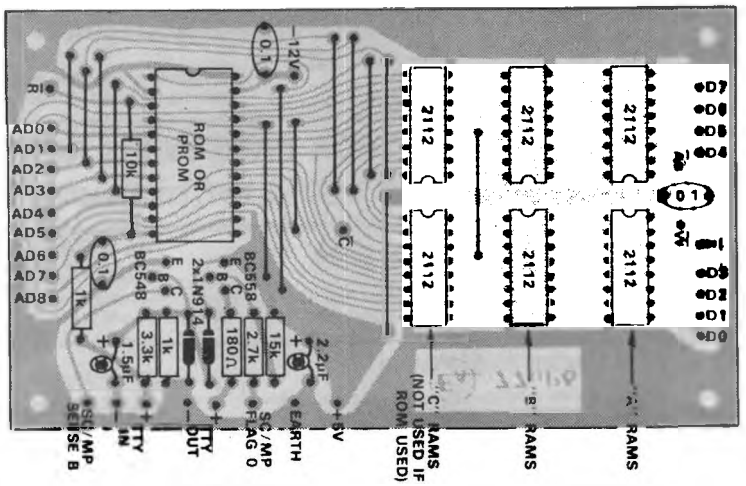
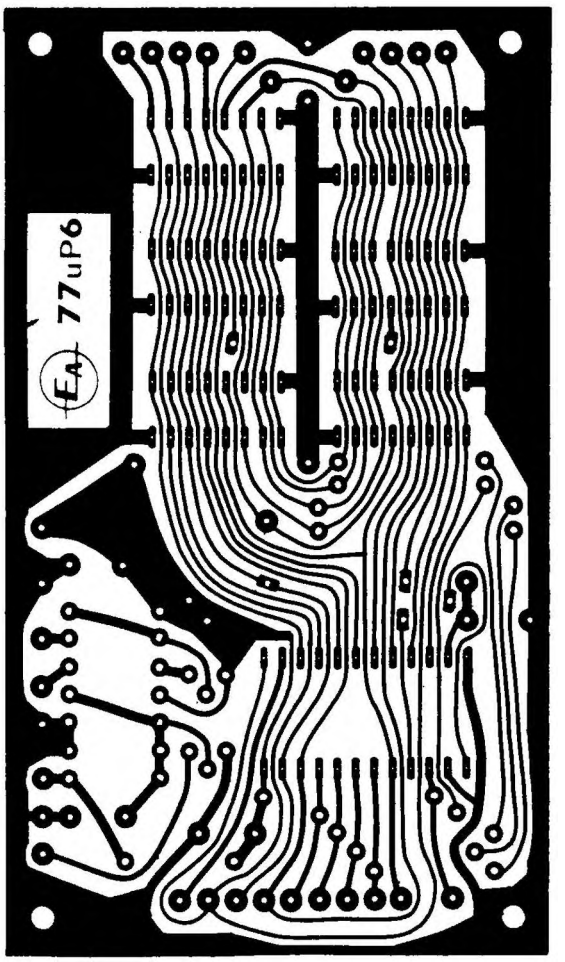


FIG. 3 DECODING FOR RAM AND ROM/PROM SYSTEM (768 BYTES RAM, 512 BYTES ROM)
WHEN ROM IS INACTIVE RAMS OCCUPY 000-2FF
WHEN ROM IS ACTIVE ROM OCCUPIES 000-1FF, RAMS OCCUPY 200-3FF, 700-7FF





At right is the PCB pattern for our add-on memory and interfacing board, shown actual size. The diagram above shows how to wire it up for both RAM and ROM, but note that only two extra pairs of RAMs can be used if you fit a ROM or PROM as well.

which can be used to print out 8-bit numbers in hexadecimal, called PHEX. This is also handy.

And in addition Kitbug provides you with some basic monitor-debug functions, so that you can feed in programs from the terminal keyboard, examine and modify them, and run them. So that it can be very convenient to have Kitbug resident in your Mini Scamp, in a ROM, so that it knows how to do these things right from switch-on.

It is fairly easy to add an MM5214 ROM or an MM5204 UV-erasable PROM to the Mini Scamp, with up to a total of 768 bytes of RAM (i.e., ¾k) as well if desired. The basic circuit is shown in Fig. 3.

As you can see some address switching is required, because SC/MP programs normally start at the bottom of memory space. Hence to run the ROM program it must be switched down in place of the existing RAM.

Although this could be done with fixed wiring, it would then not be possible to run programs in RAM under control of the front-panel switches. They could only be run under the control of the ROM monitor, using a terminal.

By using switching as shown, you are able to disable the ROM at will, and swing RAMs down into the lowest part of memory space to allow programs to be run in RAM via the front panel.

The reason for the diodes is that a 5214 ROM or a 5204 PROM is 512 bytes, so that it occupies two 256-byte memory blocks. When active it must therefore be enabled by both the 0 and 1 outputs of the 74LS138. The diodes achieve this by allowing the R-bar line to be pulled low by either decoder output.

Switch pole S3a is used to swing the original pair of RAMs up to the 300-3FF block of memory space when the ROM is in use. Similarly switch section S3c is used to swing the "A" pair of additional RAMs out of the 100-1FF block, as this is also occupied by the ROM.

Note that Fig. 3 shows the A-bar line as being switched to the "7" output of the decoder in the ROM position, so that the "A" RAMs are swung up to the 700-7FF block of memory space. This is strictly only necessary for the Kitbug ROM, which expects its RAM stack to be at the top of the current memory page.

If you are using some other ROM or PROM, you could elect to switch the "A" pair of RAMs to some other block of memory space by using one of the other decoder outputs instead. If you were to use the "4" output, for example, they would occupy the range 400-4FF when the ROM is active.

When you are using a ROM or PROM, you can only use the simple expansion technique of Fig. 3 to expand the RAM part of the system to a total of 768 bytes—¾k. This is true even when the ROM or PROM is switched out of operation using S3.

The limitation is again one of SC/MP loading, on the data and address lines.

You could of course fit sockets in the "C" RAM positions, and also in the ROM/PROM position. This would allow you to unplug the ROM or PROM when it was not in use, and plug in the third pair of additional RAMs instead. While slightly messy, this would be fairly practical if you took care in handling the devices.

Even without this trick you can still have a system with up to 1280 bytes of memory—768 bytes of RAM, and 512 bytes of ROM/PROM. This is quite a respectable memory.

To help you in expanding your Mini Scamp along the lines we have discussed so far, we have produced a small add-on PC board pattern. Coded 77uP6, the PCB measures only 130 × 75mm. However it lets you provide both the terminal interfacing of Fig. 1 and either the RAM-only expansion of Fig. 2, or the mixed RAM-ROM system of Fig. 3.

We are reproducing here the PCB pattern, actual size, for those who may wish to make their own boards. However, boards made to the pattern should be available from the usual suppliers shortly.

The wiring of the PCB is as shown in the overlay diagram. As you can see, it is fairly straightforward.

If you are planning to use the Kitbug ROM, however, there are a couple of modifications which you will have to make to the main Mini Scamp PC board. These are necessary because Kitbug uses page wrap-around addressing to establish its stack in RAM. This means that it addresses the stack as if its top location were at hex FFF—a "non-existent" memory location even on the original SC/MP kit.

The writers of Kitbug were able to take this liberty because of the simplified addressing circuitry on the original SC/MP kit. In effect, the one pair of

RAMs occupied a number of different blocks in memory space—including both the 200-2FF block and the F00-FFF block.

To allow Kitbug to operate correctly with Mini Scamp, it is necessary to pull a similar trick. In fact the easiest way is to disconnect address line AD11 from the 74LS138 decoder, so that it can no longer distinguish between the first and second 2k blocks of memory. This then allows the "A" RAMs, connected to the 7 output of the decoder, to act as if they are in memory block F00-FFF as well as block 700-7FF.

Of course the other memory blocks become duplicated too, so that the original RAMs will occupy B00-BFF as well as 300-3FF, and the "B" RAMs will occupy A00-AFF as well as 200-2FF. The ROM will also occupy 800-9FF as well as 000-1FF. But these addressing ambiguities will not cause problems, because there are no other devices at the duplicated memory blocks.

To make this modification, you will have to cut the copper track of the AD11 address line, either side of pin 4 of the 74LS138 decoder. Pin 4 of the IC can then be grounded, by linking it across to pin 5. Another insulated link should then be used to rejoin the two severed halves of the AD11 line to each other.

Even with this modification there is still a further complication if you want to use the Kitbug ROM. The simple addressing used in Mini Scamp for the front panel LEDs and switches is no longer adequate, because it clashes with that of Kitbug itself.

In other words, it is necessary to change the addressing of the LEDs and switches. The easiest way of doing this is shown in Fig. 4.

As you can see, it involves removing the original link used to gate G2 and G3 from the AD11 line, and using one of the originally "spare" 74LS05 inverters to gate them instead from the unused "4" output of the 74LS138 decoder.

This is fairly easily done, and involves only an extra 10k resistor and two short lengths of hookup wire.

Note that with this modification done, the front panel LEDs will now have the hex address 402, instead of the original address 802. Similarly the switches will now have the address 401 instead of 801.

If you don't remember this, you may find that programs written for Mini Scamp in its original form won't work! I found this out the hard way, myself. For example with both of Dr. Kennewell's sample programs given in the original Mini Scamp article, you will need to change the first real instruction to LDI 04 (hex code C404), before they will work.

Well, that's the story on how to expand your Mini Scamp easily into quite a respectable little machine. Perhaps there's only one more question to answer: is it possible to expand the system still further?

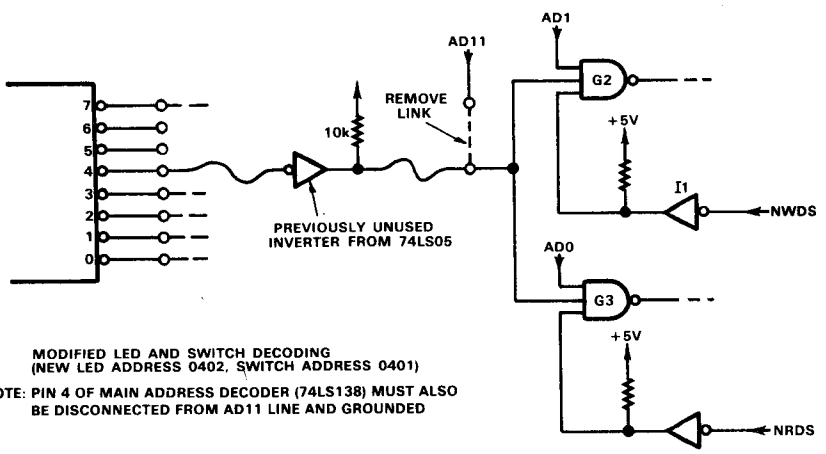
The answer to this is yes, but with qualifications. To add further memory, you will need to use a more thoroughgoing approach. The address lines will need to be buffered, while the data lines will have to be provided with bidirectional bus transceivers. Similarly the address decoding will need to be extended, to provide decoding for more blocks of memory.

It will probably also be necessary to buffer the Write strobe line (NWDS), at least. And of course you will need a suitable PC board to mount the additional memory devices, so there will be the business of mounting all of this additional circuitry in the Mini Scamp case.

While you could probably do all this, you would really be stretching the Mini Scamp concept beyond its normal bounds. The end result would start to become a mechanical monster, and rather hard to troubleshoot if anything goes wrong.

If you really do want to expand to a big system, the best plan might be to change over to a modular system rather than attempt to enlarge the Mini Scamp concept still further.

Most of the parts you have used in Mini Scamp should be suitable for use in a modular system, so that with a little care you should be able to make the change at relatively low cost.



MODIFIED LED AND SWITCH DECODING
(NEW LED ADDRESS 0402, SWITCH ADDRESS 0401)

NOTE: PIN 4 OF MAIN ADDRESS DECODER (74LS138) MUST ALSO BE DISCONNECTED FROM AD11 LINE AND GROUNDED

FIG. 4 MODIFICATIONS FOR KITBUG ROM

More on Mini Scamp

Judging by the tremendous interest it has generated already, our Mini Scamp looks like becoming the most popular microcomputer project ever described. This article gives details of some simple modifications and improvements to the basic design, to improve performance. It also discusses simple techniques for providing parallel interfacing.

by JAMIESON ROWE

One good thing about a popular project like Dr. Kennewell's Mini Scamp is that with so many people building it up quickly, any errors or bugs present tend to show up sooner than otherwise!

For example, the May issue had only been published a couple of days before a reader rang to point out that there was an error in the power supply wiring diagram.

If you didn't see our note last month about that error, it showed the wire from the 12.6V transformer tap as being connected to earth. In fact it should only connect to the "+" end of the 1000uF electrolytic capacitor, of course.

It took a little longer for any other bugs to show up, but after a couple of weeks we received reports of some builders having trouble with the deposit timing circuitry. Some units wouldn't load in programs properly in DMA mode, while others wouldn't load from the switches reliably under program control.

When we looked into this, we found a subtle timing problem associated with the deposit timing delay circuit shown in the May issue. With some 1N914 diodes and BC108 transistors, charge storage times were sufficient to slow down the leading edge of the pulse used to gate the data switches onto the data lines, and improper loading occurred. Usually, some or all of the bits loaded as "1's", regardless of the switch positions.

Happily, we found a simple way of fixing this trouble. All that is required is to reduce the value of the base pulldown resistor shown as 10k, to 680 ohms. This should produce reliable loading with all devices.

In the meantime, Mini Scamp designer Dr John Kennewell advised us that he had found it desirable to add a 180-ohm resistor in series with the deposit switch. This prevents the switch from taking the B1 input of the 74123 monostable right up to the 5V rail, and thus prevents the monostable from being spuriously triggered by supply line transients.

Incidentally you may have noticed that while Dr Kennewell's original circuit as shown in the April article shows a DPDT switch used for S1, performing the DMA/CPU switching, we called for only an SPDT switch in the parts list. This is because the SPDT switch is quite capable of doing the job, if its rotor is taken to earth. In the DMA position it earths the two wires from the PCB pads marked "DMA/CPU SW A", while in the CPU position it earths the single wire from the PCB pad marked "DMA/CPU SW B". Most builders seem to have deduced this for themselves, but I mention it in case you haven't got that far as yet.

If you make these few minor corrections to your Mini Scamp, you should find that it works quite well. However something which may become apparent once you begin running programs is that with the Mini Scamp as it stands, the LOAD SWITCHES instruction tends to result in an automatic STORE LEDS operation, whether you want this to happen or not.

This occurs because of the 1N914 diode tying the NWDS control line to the Q2-bar output of the 74123 monostable. The diode is used for DMA program loading, serving no useful purpose in CPU mode. However as it is still in circuit, it is able to cause spurious writing into the 74C175 LED latches, when the deposit switch is pressed as part of a LOAD SWITCHES instruction.

Again, there is a fairly simple way of preventing this from happening. All that is required is to replace the existing SPDT switch used for DMA/CPU selection, with a DPDT type. The second section of the new switch is then used to break the connection between the 1N914 diode and the NWDS line, in the CPU position.

You don't need to cut any PCB tracks to make this change, because the diode is normally connected to the NWDS line via a wire link. All you need do is cut the link, and connect the two cut ends to the appropriate lugs of the new DMA/CPU switch.

There is another modification to the basic Mini Scamp design which some builders may care to make. Although it requires a little more work, it is still fairly straightforward and involves only a handful of additional low-cost parts.

The purpose of this further modification is to ensure that only a single loading takes place from the switches when the deposit switch is pressed in CPU mode, in response to a LOAD SWITCHES instruction having lit the DRQ LED. With the existing Mini Scamp circuit multiple

loading can occur if the CPU executes another LOAD SWITCHES instruction while the deposit switch is still pressed, as the 7476 DRQ latch can re-trigger the 74123 via inverter I4. This is a side effect, because I4 is in circuit mainly to ensure that the 74123 can only be triggered in CPU mode when the DRQ latch has been set for a LOAD SWITCHES instruction. If the 74123 were to be triggered at other times, a program could be changed and caused to "crash".

The only way of preventing multiple loading with the circuit as it stands is to insert delay instructions into your program, so that one has time to release the deposit button before the CPU can get to the next LOAD SWITCHES instruction. This is not always convenient.

To obviate the multiple loading, I4 can be used to gate the 74123 by means of its clear input, instead of the A1 trigger input. This is done as shown in Fig. 1.

Note that the A1 input of the 74123 is now earthed directly, with the output of I4 now taken to the active-low clear input on pin 3. As I4 is an open-collector element, a 10k pullup resistor must be added as shown.

Diode D1 from the DMA/CPU switch is now taken to the input of I4, so that the 74123 can be enabled in DMA mode as before. The input of I4 is disconnected from the Q input of the 7476 DRQ latch, and connected to the Q-bar output via diode D2 to achieve the correct logic action. A further 10k resistor is used at the input of I4, to form a single OR gate with the two diodes.

Although this fixes the multiple loading problem, one can still get occasional faulty loads from the switches. This seems to be due to spurious triggering of the 74123 from the deposit switch, as a result of the low slope produced by the simple R-C bounce integrator.

Presumably the input of the 74123 can become unstable during the transition from logic low to logic high levels.

To fix this we suggest that you replace the deposit switch with a SPDT type, and use the originally redundant second half of the 7476 device as a debounce latch, as shown in Fig.1. This gives completely reliable operation.

Note that Fig. 1 also shows the optional switching between diode D3 and the NWDS line, and the 680 ohm base resistor in the deposit timing delay circuit.

To make these suggested modifications, you need to cut a small number of PCB tracks and add a few links under the PC board. We suggest you do this in the following order, to make sure that you don't lose track of anything.

1. change the 10k resistor at the base of the BC108 used to delay the 74123 Q-bar output signal fed to the data switch control gates to 680 ohms.
2. cut the track between pin 1 of the 74123 and pin 10 of the 74LS05.
3. cut the track between pin 10 of the 74LS05 and the anode of D1.
4. cut the track linking pin 11 of the 74LS05 to the track joining pin 13 of the 74LS05 and pin 15 of the 7476.
5. remove the 470 ohm resistor and 47uF tantalum electrolytic capacitor connected to PCB point "DEP".
6. unsolder the wire connecting the deposit switch to the PCB point "DEP", at the deposit switch.
7. replace the deposit switch with a momentary contact SPDT push-button (C&K type 8121 or similar).
8. connect the NO and NC contacts of the deposit switch to +5V rail with two 10k resistors (at the deposit switch).
9. connect the common contact of the deposit switch to GND.
10. drill two holes near pins 7 and 8 of the 7476, and connect these pins to the NO and NC contacts of the deposit switch respectively.
11. drill a hole near pin 11 of the 7476, and connect the loose wire attached to PCB point "DEP" to pin 11 of the 7476.
12. connect pin 3 of the 74123 to pin 10 of the 74LS05.
13. connect pin 1 of the 74123 to pin 8 of the 74123 (i.e., earth).
14. connect the anode of D1 to pin 11 of the 74LS05.
15. connect a 10k resistor from the anode of D1 to the +5V rail (pin 14 of the 74LS05).
16. connect the anode of additional diode D2 to pin 11 of the 74LS05, and the cathode to pin 14 of the 7476.
17. connect a 10k resistor from pin 10 of the 74LS05 to the +5V rail (pin 14 of the 74LS05).
18. join pins 2, 4, 6, 9, 12 and 16 of the 7476 to pin 11 of the 74123, and then connect them to the +5V rail via a 1k resistor.

With these modifications, your Mini Scamp should leave nothing to be desired in terms of its internal operation. But perhaps a few words are now in order for the benefit of those readers who are planning to use their Mini Scamp to control external devices.

For simple applications, you may be able to use direct interfacing to the flag and sense lines of the SC/MP chip itself, rather like that used for the serial interface shown last month. The SC/MP chip has three flag outputs and two sense inputs, and also two other pins designated "serial in" and "serial out". This allows for a total of seven input-output lines.

Fig. 2 shows how one of the flag lines or the SOUT line could be used to control a small relay. It also shows how a SPDT switch can be debounced and fed to one of the sense inputs, or the SIN input.

For more complex applications, you may wish to provide Mini Scamp with full 8-bit parallel interface ports. This can be done fairly simply, providing you can write your program so that it isn't worried

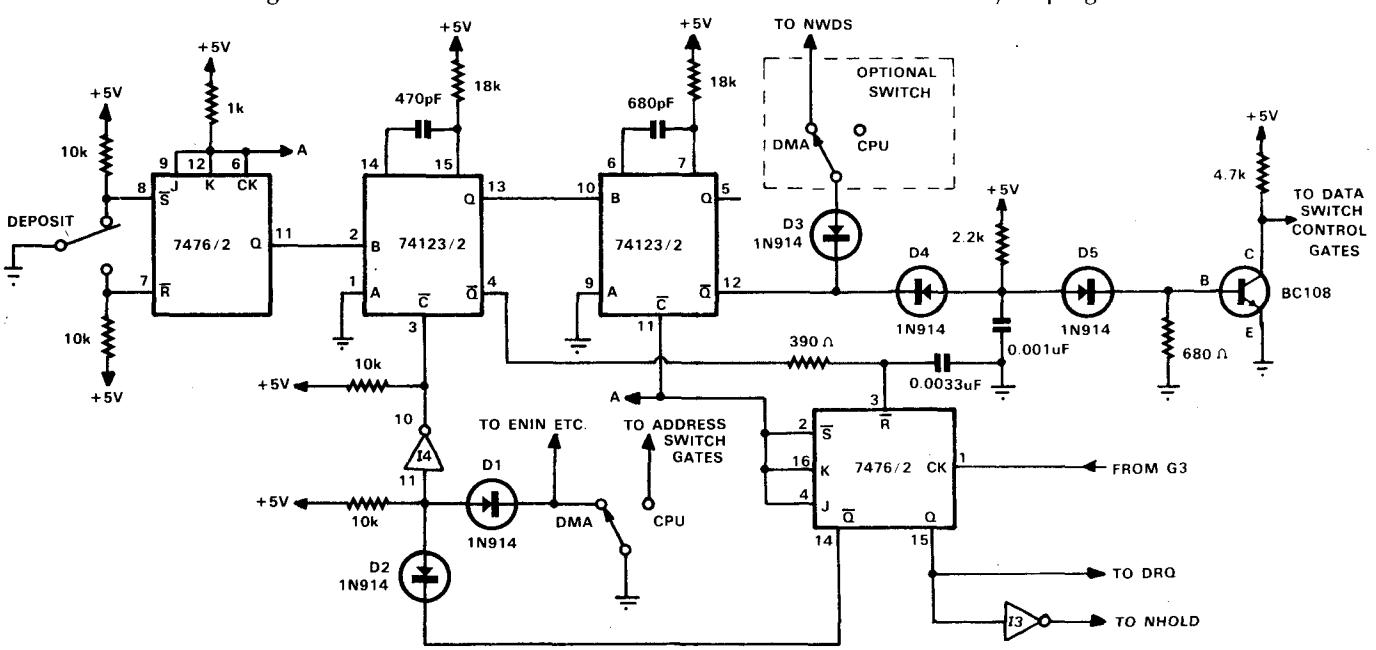


FIG. 1: MODIFIED MINISCAMP DEPOSIT SWITCH LOADING LOGIC

File No. 8/M/—

MINI SCAMP

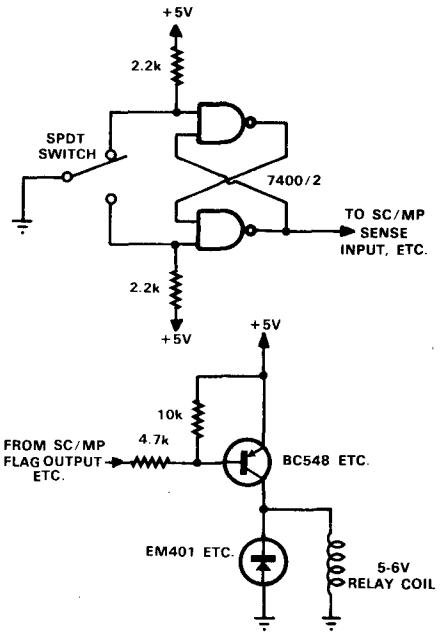


FIG. 2: SIMPLE RELAY & SWITCH INTERFACING

The three diagrams above and at right should give you some guidance on providing your Mini Scamp with interfacing to the outside world.

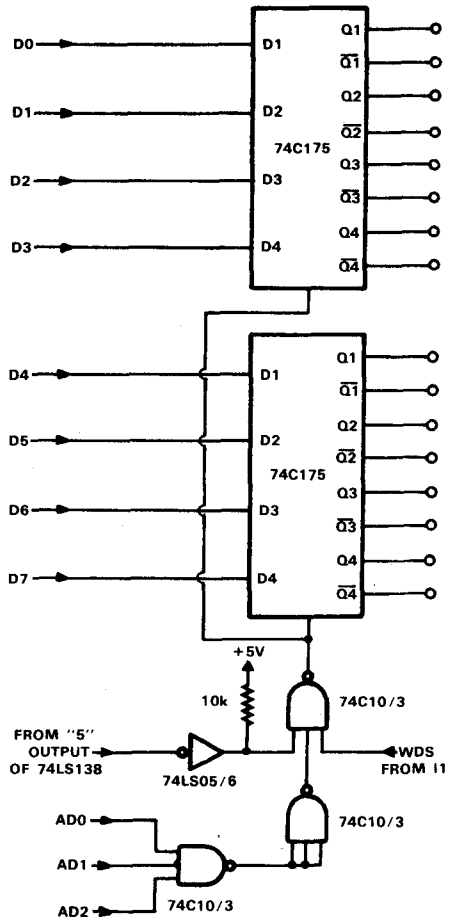


FIG. 3: SIMPLE LATCHED 8-BIT OUTPUT PORT (ADDRESS 507 HEX)

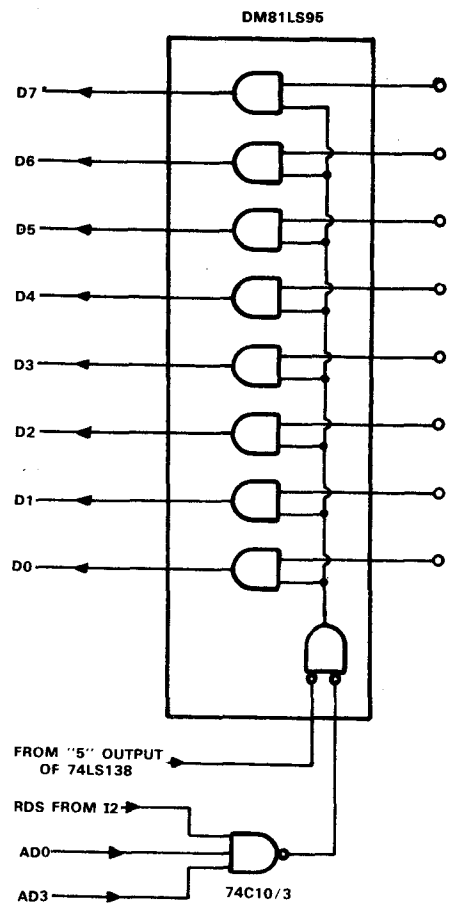


FIG. 4: SIMPLE 8-BIT INPUT PORT (ADDRESS 511 HEX)

by the ambiguous addressing which results from incomplete decoding.

A simple latched 8-bit output port is shown in Fig. 3. Two 74C175 devices are used for the latch itself, providing both active-low and active-high outputs for the eight bits. These outputs could be used to drive external circuits via buffers like those used for driving the LEDs in Mini Scamp itself.

As you can see, the address decoding is quite simple, combining the "5" output from the 74LS138 decoder with the AD0, AD1 and AD2 address line signals to give the port an effective address of 507 hex. When a STORE instruction specifying this address is executed, the WDS pulse from I1 is allowed to reach the clock inputs of the latches, writing the data into them.

Because the gating does not sense address lines 3, 4, 5, 6 or 7, the port effectively occupies other addresses besides 507. For example it could also be addressed as 50F, 517, 51F, 527 and so on. This shouldn't cause any strife as long as you remember it when writing your programs.

The same sort of simplified addressing is used in the 8-bit input port shown in Fig. 4. Here only a single 74C10 gate element is used, in conjunction with an internal two-input enabling gate provided in the DM81LS95 octal buffer. RDS pulses from inverter 12 are only effective in enabling the buffer when address lines AD0 and AD3 are high and the "5" output of the 74LS138 decoder is low, corresponding to address 511 hex.

Hence you can read the data from the port into the SC/MP accumulator by executing a LOAD instruction which references address 511. But because of the simple decoding, you can also regard the port as having address 513, 515, 517, 51B, 531 and so on.

Both of the circuits shown in Fig. 3 and 4 impose very low loading on the lines SC/MP address, data and control lines so you should be able to use them with Mini Scamp even if you have expanded the memory. However if you want to add a number of different ports along these lines, you may have to add extra buffering to ensure reliable operation.

NO LOADING IN CPU MODE?

A small number of readers have found that while their deposit switch functions correctly in DMA mode, it will not operate in CPU mode. This appears to be due to excessive voltage drop across the 390 ohm resistor linking the Q1-bar output of the 74123 (pin 4) to the reset input of the 7476 DRQ latch (pin 3). This prevents the DRQ latch from being reset when the deposit switch has been operated.

The cure is to reduce the resistor to 220 ohms and increase the value of the associated capacitor from 0.0033uF to 0.0068uF.

A "baby" system using the Signetics 2650

Here is surely the simplest and lowest-cost way of getting to know the Signetics 2650 microprocessor. A complete microcomputer system on a single small PC board, you can build it for around $70, not counting a power supply or terminal. Despite its low cost, it offers the same debug and monitor program in ROM provided by more expensive systems, together with 256 words of RAM.

by JAMIESON ROWE

As we have noted in earlier articles, the Signetics 2650 microprocessor is a particularly powerful device. Its architecture and instruction set are very minicomputer-like, making it well suited for general-purpose computing as well as low-cost dedicated applications.

In their literature, Signetics note that the device may be used to implement a very low cost minimal "evaluation kit" type system, one which would be very suitable for those wishing to gain experience with the 2650 with the minimum outlay of both time and money. However they themselves have not made such a minimal evaluation system available, only larger systems such as the PC1001 and PC1500 systems.

This seemed rather a pity to us, as at least one other microprocessor has been available in a really minimal system, and this has proved very popular. However as the 2650 and its 2608 ROM chip have been in rather short supply until recently, there seemed little hope of being able to remedy the situation as far as the 2650 was concerned.

Happily this situation has now changed for the better. Just a few weeks ago we learned from Philips Industries that the 2650 and 2608 chips were now readily available, and at relatively low cost. (Signetics is a US subsidiary of Philips.) We accordingly suggested to them that this would be an ideal opportunity to produce a low-cost "baby" 2650 system, based on the minimal system suggested by Signetics themselves. They agreed, and offered to make available a set of devices if we cared to try the idea.

This project is the result!

Basically, it is a complete general-purpose microcomputer, just like the larger evaluation kits. In fact it has the same debug and monitor program as the larger kits—"PIPBUG"—resident in the 2608 ROM (1k x 8-bit words). It communicates directly with a standard 20mA asynchronous data terminal, such as an ASR-33 Teletype or the video data terminal described in our January and February issues, and requires a single 5V DC power supply.

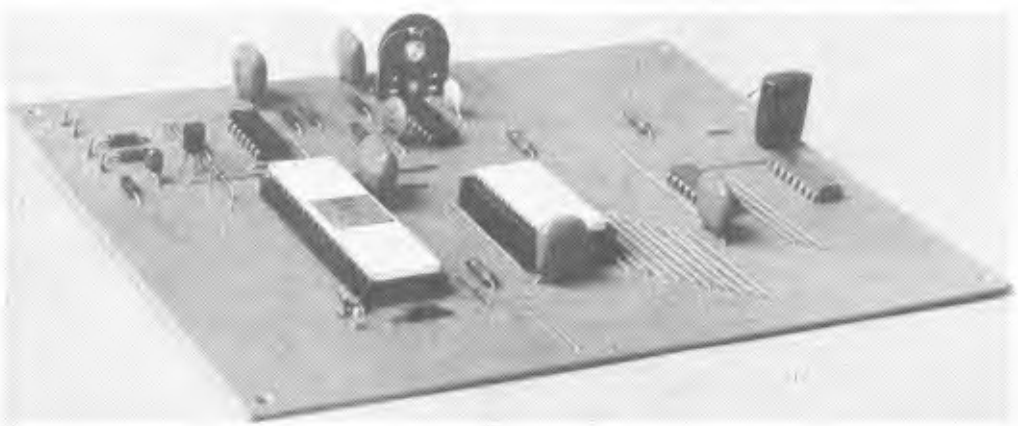
The main difference between this system and the larger systems is that there is only 256 bytes of RAM memory for user program storage, and there is no on-board decoding or buffering for further memory or peripheral expansion.

In short, it is a "bare minimum" 2650 system, designed to be the cheapest and easiest way of getting a 2650 up and running. At the same time, it offers the full program development facilities of PIPBUG, including the ability to examine and alter memory from the terminal keyboard; the ability to dump programs to paper tape or cassette, and then load memory from tape; the ability to examine and set the processor registers; the ability to set and remove up to two breakpoints, for debugging; and the ability to run the user's program on command.

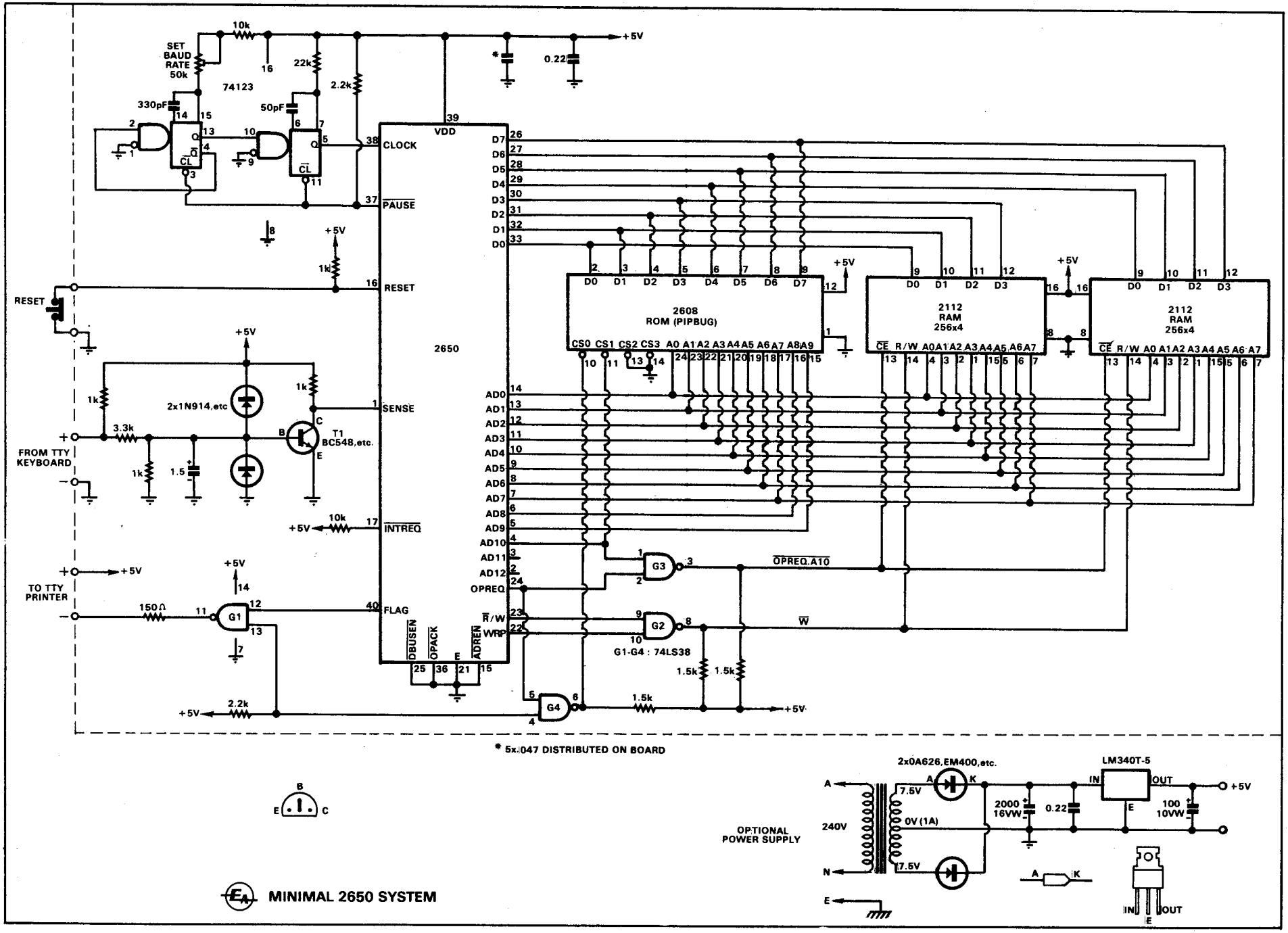
These are quite powerful program development facilities, not usually found on low cost systems. As a result, our "baby" 2650 microcomputer should be particularly suitable for educational and training purposes, whether by schools and colleges or by individual enthusiasts.

As you can see from the diagrams and photograph, it consists of only a handful of parts on a small PC board. There are only six ICs, one transistor and a few resistors and capacitors, and the PCB is single-sided to keep the cost down.

Heart of the circuit is the 2650 chip itself, a powerful 8-bit microprocessor with an instruction set of 75 instructions, and eight different addressing modes. Fabricated using low-threshold ion implantation, it is an N-channel silicon



At left is our new "baby" 2650 microcomputer, complete on its small PC board. It offers the same PIPBUG program in ROM as provided on the larger systems. The full circuit diagram is shown on the facing page, together with an optional power supply.



MINIMAL 2650 SYSTEM

gate device which operates from a single 5V supply and offers TTL compatibility on all inputs and outputs.

A 74123 dual monostable device is used to generate the single-phase 1MHz clock signals for the 2650. The clock oscillator is of the R-C type, but is easily adjusted to the correct operating frequency without the need for elaborate instruments. More about this later on . . .

As mentioned already, the PIPBUG debug-monitor program is resident in a 2608 ROM. This includes routines for servicing the data terminal input and output, so that the system "knows" how to communicate with a terminal as soon as it is initialised. The code suffix for the 2608 with PIPBUG resident is CN0035.

Two 2112 devices are used to provide the RAM memory of the system. These are low-cost static MOS RAMs, each organised as 256 words or 4 bits, so that the two together provide a RAM of 256 8-bit words. Some 63 words are used by PIPBUG as its scratchpad area, leaving 193 available for user programs.

The remaining IC in the circuit is a 74LS38 low-power Schottky quad NAND buffer, two gates of which are used for simple address decoding to allow the 2650 to differentiate between the ROM and RAM sections of memory.

The ROM is allocated to the address range 000—3FF hexadecimal, or the first 1k of memory space. The RAM memory is allocated to the next 256 bytes of memory space, with hexadecimal addresses 400—4FF. Basically this means that when binary address bit AD10 is 0, the ROM is selected, while when it is 1 the RAM memory is selected.

Gate G3 is used to enable the two 2112 RAM devices when AD10 is at the 1 level. The second input of G3 is fed with the OPREQ signal from the 2650, which is a strobing signal used to indicate when bus information is valid.

When AD10 is at the 0 level, the RAMs are therefore disabled. At the same time the ROM is enabled, because the AD10 signal from the 2650 is also fed to the active-low chip-select input CS1 of the 2608 ROM device. Correct strobing of the ROM is achieved by using gate G4 as an inverter to feed an OPREQ-bar signal to the CSO input of the ROM.

Note that this simple address decoding scheme is not completely unambiguous, because the ROM is enabled whenever AD10 is 0 and the RAMs whenever it is 1. Thus the ROM strictly occupies not only the nominal range of 000-3FF, but also higher ranges such as 800-BFF. Similarly the RAMs occupy not only their nominal range 400-4FF, but also higher ranges such as 500-5FF, 600-6FF, 700-7FF, C00-CFF, D00-DFF, E00-EFF, and F00-FFF.

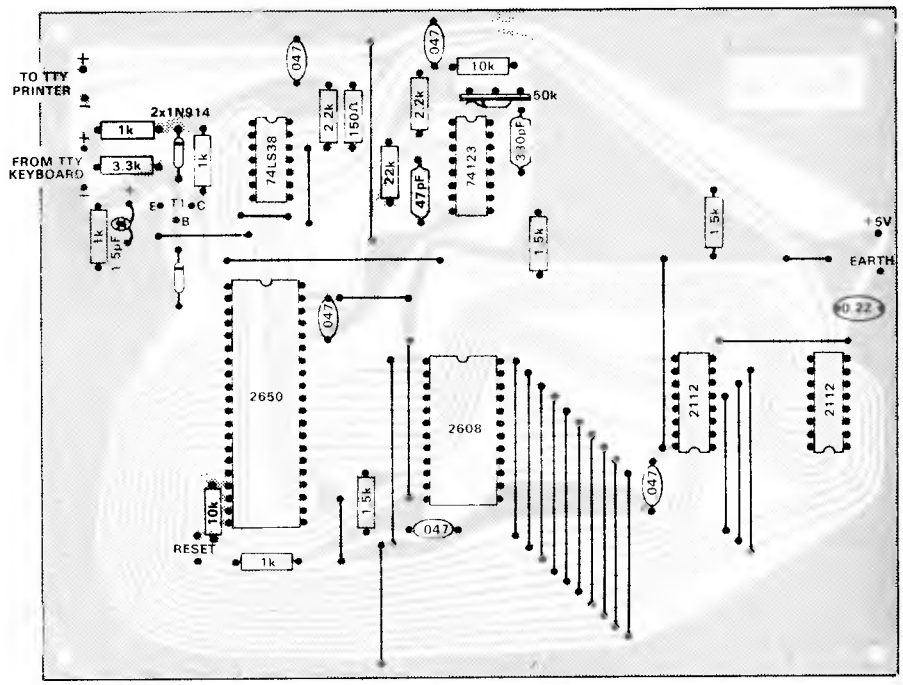
This ambiguity need not cause any complications, however, providing you remember it and take it into account when writing your programs. All it means is that if you forget and your program tries to address these non-existent higher memory locations, it will in reality still be talking to the same ROM and RAMs!

Many small systems use this type of simple address decoding, to simplify the circuitry and reduce costs.

The third gate, G2, is used to control the read-write function selection of the RAM devices. The inputs of the gate are fed from the R-bar/W and WRP outputs of the 2650, while its output goes to the R/W-bar control inputs of the 2112 RAM devices. The R-bar/W output of the 2650 provides its read-write control signal, while the WRP output provides a write strobe pulse designed to delay writing until memory devices have stablised.

The remaining section of the circuit is that used to provide the serial communication ports, which are associated with the flag (F) output and sense (S) input of the 2650. The output port uses the remaining gate G1 as a buffer, to control a 20mA output current in response to the F output of the microprocessor. The 150-ohm resistor in series with the gate output sets the output current level, which is sufficient to drive the normal current-loop input of an ASCII teleprinter or video data terminal.

The input port circuitry uses a BC548 or similar general-purpose NPN transistor T1 to provide level translation between a standard 20mA current loop input and the S input of the microprocessor. The input circuit provides its own 20mA source, and so is suitable for direct connection to the keyboard contacts of a teleprinter, or the corresponding output terminals of a video data terminal such as that described last month.



Using this overlay diagram you should have no problems in fitting the components to the PC board. Sockets are used for the 2650 and 2608 devices.

LIST OF PARTS

1 PC board, 175 x 135mm, code 77up2
8 PCB terminal pins (optional)
1 2650 microprocessor IC
1 2608 ROM (with PIPBUG: code CN0035)
2 2112B RAMs
1 74123 dual monostable
1 74LS38 low-power Schottky buffer
1 BC548 or similar NPN silicon
1 1N914, 1N1418 or similar diodes
1 40-pin IC socket, PC type
1 24-pin IC socket, PC type

RESISTORS

Quarter watt, 5%: 150ohms, 4 x 1k, 3 x 1.5k, 2 x 2.2k, 1 x 3.3k, 2 x 10k, 1 x 22k.
1 47k PC type tab pot, vertical mount

CAPACITORS

1 47pF NPO ceramic
1 330pF NPO ceramic
5 .047uF LV polyester
1 0.22uF LV polyester
1 1.5uF 35VW electro or tantalum
Wire for links, solder, etc.

The 1.5uF capacitor in the base circuit of T1 is to provide contact bounce suppression in the case of teleprinter keyboards, and also to provide filtering of any noise induced in the input line. The two diodes are to protect the transistor from high amplitude noise impulses.

As you can see, the complete baby system is built up on a small PC board measuring 175 x 135mm. The pattern is coded 77up2, and PC boards etched to the pattern should be available from board manufacturers by the time you read this article.

Assembly of the system on the PCB should be fairly straightforward using the overlay diagram as a guide. Note that there are a number of wire links, necessary because the board has been kept single-sided.

In view of the fact that the 2650 microprocessor chip and the 2608 ROM are both fairly expensive, and are both MOS devices, I suggest that you use sockets for them. A 40-pin socket is required for the 2650, and a 24-pin socket for the 2608, both being of the 0.6in row-spacing DIL type. Use high quality sockets if you can, to avoid contact troubles.

Below is the PCB pattern, actual size for those wishing to etch their own.

The remaining ICs are probably best soldered directly into the PC board.

I suggest that you wire in all of the links first, then add the IC sockets and the resistors and capacitors. Watch the polarity of the 1.5uF tantalum electrolytic, as this could cause malfunction if it is connected the wrong way around.

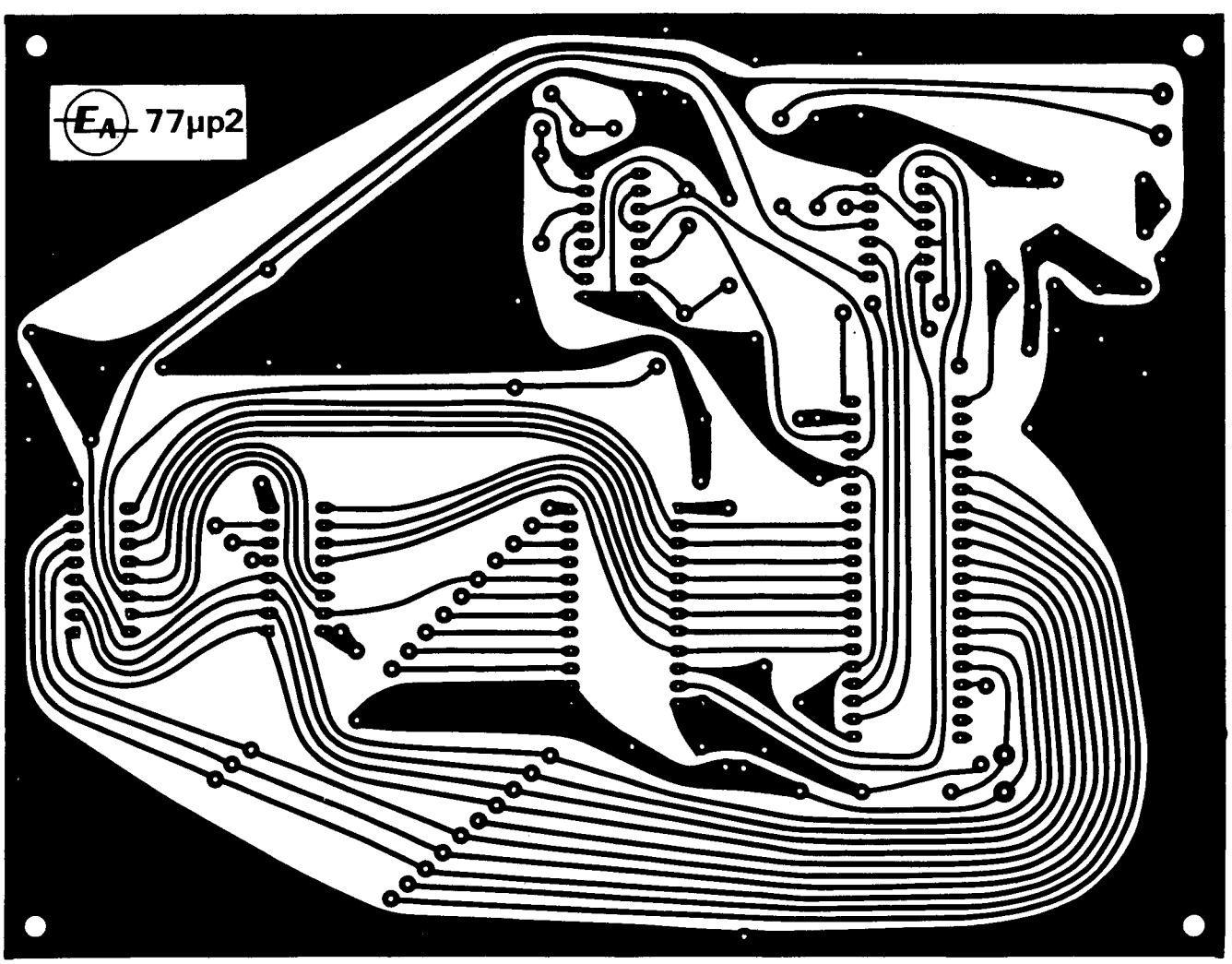
Now wire in the transistor, the two diodes and the two TTL ICs (74123 and 74LS38), taking care that these are also orientated correctly. Then finally add the two RAMs, after having connected the barrel and bit of your soldering iron to the PCB supply lines to ensure that the MOS ICs won't be damaged by static charge. It is a good idea to solder the supply pins of each IC first (pins 7 and 14), so that the internal protection diodes become operational as soon as possible.

There are only eight external connections to the PC board. Two are for the power supply, which may be almost any reasonably well regulated and filtered 5V DC supply. The total current drain is around 250mA. If you don't have a suitable supply handy, the circuit shown in the small diagram would be quite suitable.

The four connection points adjacent to one another are for the serial input and output. These connect to the teleprinter or video data terminal, with polarities as shown. Whichever type of terminal is used, it should be connected for 20mA, full duplex operation.

The remaining two connections to the PCB go to the reset switch, which is a simple normally-closed pushbutton. When pressed, this button forces the microprocessor to reset its internal registers. Then when the button is released the microprocessor begins running from a known and predetermined state, fetching its first instruction from memory location 000—the start of the PIPBUG program residing in ROM.

The reset button therefore serves to initialise the system, and is used for this purpose both when power is first applied, and at other occasions whenever one wishes to return to PIPBUG from an applications program (apart from breakpoint returns, which take place automatically).



When you have completed wiring the PC board and connected it up to the terminal, reset switch and power supply, carefully remove the 2650 and 2608 chips from their conductive foam and plug them into their respective sockets (with the power turned off).

There is only one adjustment to be made, that in which the 74123 clock oscillator is set up to operate at the correct frequency of 1MHz. This is done with power applied.

If you have access to a frequency counter, it can be done by simply connecting the counter between pin 5 of the 74123 and the grounded negative supply rail, and adjusting the small tab pot until the counter reads 1MHz. This is the preferred way of setting the clock frequency.

However if you don't have access to a counter, the frequency can still be set up fairly accurately using the teleprinter or data terminal itself. This can be done because only when the clock frequency is the correct 1MHz will the software-timed serial output signals produced by the 2650 be at the correct 110-baud data rate required by the terminal.

To adjust the clock frequency using this method, apply power to both the system and the terminal. Then press the reset button, and release. The printer of the Teletype or the screen of the video terminal should print a couple of characters and then become static.

If by some lucky chance you have the correct clock frequency already, the printer or display screen should have displayed a carriage return (CR), a line feed (LF), and an asterisk. This is the programmed output of the PIPBUG program upon initialisation (the asterisk is its prompt signal, signifying readyness for an input command).

Most likely, you won't get this sequence of CR-LF-asterisk straight away. But the idea is to adjust the tab pot slowly and carefully from its maximum resistance extreme, pressing the reset switch after each change until you find the setting where the terminal shows that the characters are being fed to it at the correct rate.

There should be a small zone of the pot's travel in which the characters are printed correctly following the release of the reset button. For most reliable operation, try to set the pot in the middle of this zone.

With this adjustment made, your baby 2650 system is fully operational and ready to begin work (or play!). With the set of ICs, you should have received a Signetics Applications Memo (coded SS50) which explains how to use PIPBUG to feed applications programs into the system, run them, debug them, dump them on paper tape (or cassette), and re-load them. It also gives a listing of PIPBUG itself, which among other things lets you make use of some of its utility subroutines such as the serial input and output routines "CHIN" and "COUT".

```
SIMPLE ANSWER-BACK PROGRAM FOR "BABY" 2650 MICROCOMPUTER
WRITTEN BY J.ROWE, "ELECTRONICS AUSTRALIA" MAGAZINE

ADD.  CODE      MNEMONICS          COMMENTS

440   76 C0     PPSU   40          /SET TTY TO MARK
442   3F 02 36  BSTA,UN CHIN       /FETCH CHAR VIA PIPBUG RTN
445   C1        STRZ,R1            /SAVE
446   3F 02 B4  BSTA,UN COUT       /ECHO
449   01        LODZ,R1            /RESTORE IN R0
44A   A4 0D     SUBI,R0 "CR"       /TEST FOR CR
44C   58 74     BRNR,R0 -12        /KEEP GOING IF NOT
44E   04 0A     LODI,R0 "LF"       /IF YES, PROVIDE LF
450   3F 02 B4  BSTA,UN COUT
453   05 00     LODI,R1            /SET R1=0
455   0D 24 66  LODA,R1 466+       /FETCH ANSWER CHAR
458   C3        STRZ,R3            /SAVE
459   3F 02 B4  BSTA,UN COUT       /PRINT
45C   A7 0D     SUBI,R3 "CR"       /TEST FOR CR
45E   5B 75     BRNR,R3 -11        /IF NOT, KEEP GOING
460   04 0A     LODI,R0 "LF"       /IF YES, SUPPLY LF
462   3F 02 B4  BSTA,UN COUT
465   1B 5A     BCTR,UN -38        /BACK TO START AGAIN

467   47 4F 20                     /ANSWER TEXT
46A   41 57 41
46D   59 2C 49
470   27 4D 20
473   42 55 53
476   59 21 0D                     /ANSWER MUST END WITH A CR.
```

This simple novelty program should help you get your system going. Only the code is actually fed into RAM, starting at location 440 hex.

To help you get your system up and running, a listing is shown for a modified version of the novelty answer-back program which the author originally wrote for the larger PC1001 system. Note that all you actually enter into the system are the two-digit hexadecimal machine code words; the mnemonics and comments are purely to help follow how the program works.

To feed the program into the system, you use the PIPBUG "A" command, typing first "A 440" and then a carriage return. PIPBUG will then type out on the next line "440 XX", where XX is the current content of location 440 (probably random). It then pauses. You then type "76", followed by a line feed, whereupon PIPBUG does a CR-LF, and then prints out the next memory address and its current content. You then type "CO" and LF, and so on until all of the program has been fed in.

Then to run the program, type "G 440" followed by a CR. PIPBUG will then transfer you to the program in RAM.

Try typing in a comment, ending it with a carriage return. The program should answer with a terse "GO AWAY, I'M BUSY!"

When you get tired of this reply, it can be changed by feeding in a new string of ASCII characters starting at address 467 hex. Note, however that the message must end with a CRR (hex 0D).

Of course this is just a demonstration program, to get you going. The next step is to write your own, using as a guide the Signetics 2650 programming book supplied with the kit. Happy computing!

Baby 2650: Modifications

BABY 2650 COMPUTER (March 1977, File No. 2/CC/18): The parts list specifies only one 1N914 or similar diode, whereas two are required as shown correctly in the circuit and wiring diagrams. The quoted price of $70 for the kit does not include sales tax; when this is included the price is approximately $85. Note also that to allow better for component tolerance variations in the clock circuit, the 330pF capacitor should be changed to 270pF and the 10k resistor in series with the tab pot should be changed to 6.8k. These changes should ensure that all clocks may be set to the correct frequency of 1MHz. To improve ease of adjustment, the tab pot may also be reduced in value to 10k, or even 4.7k if desired.

Video data terminal for microcomputers

Low in cost yet loaded with features, this video terminal design is ideal for use with microcomputer systems. It uses any standard TV receiver or video monitor as the display, has an internal refresh memory capacity of 1,024 characters, and can communicate at any of nine crystal-locked data rates. It may also be used as a "TV typewriter" for titling and text generation in video systems.

by JAMIESON ROWE

Most of the small microcomputer systems which are currently arousing so much interest are designed to communicate with the user in ASCII code, via an asynchronous serial terminal such as a Teletype model ASR-33 teleprinter. The problem is that one of these teleprinters can cost you anything from $400 to $1500—rather more than many of the small microcomputers themselves!

To provide one lower-cost alternative, the author developed the ASCII-Baudot translator unit which was described in the October 1976 issue of this magazine. The translator was based on a dedicated microprocessor, and allowed a low-cost surplus Baudot teleprinter machine to communicate with a microcomputer system.

While the translator has allowed quite a few people to get their microcomputer systems "up and talking", we realised even when we were developing it that it was by no means the complete answer for everyone. Teleprinters are fairly complex mechanical devices, and surplus Baudot machines in particular can be noisy in operation and critical to adjust.

Because of the disadvantages of teleprinter machines, many people have expressed interest in a fully electronic terminal, in particular a video terminal.

Commercial video terminals have been available for some time, although until recently their cost has tended to be even higher than teleprinter machines. However, in the last year or so, low cost video terminals for microcomputers have been described in some of the US electronics and computer hobby magazines. These all used a conventional TV receiver or video monitor as the display, in order to lower costs.

Virtually all of these designs were based on particular microcomputer systems, and were intended to interface directly with the system via its parallel address and data bus lines. This generally made them unsuitable for use with other systems, particularly most of the newer microprocessor evaluation systems.

Nethertheless, they showed the way, suggesting that it should be possible to produce a similarly low-cost terminal designed for standard asynchronous serial interfacing. Accordingly, as soon as the ASCII-Baudot translator project had been completed, the author set about developing a terminal along these lines.

As luck would have it, I hadn't gone very far with a design when I received a fateful telephone call. It was from Mr Ed Monsour, a local electronics engineer who has specialised for some years now in the design of video terminals and associated equipment. He had rung to

Here is the complete terminal, with a 21-cm portable TV receiver used as the display device. The low-profile case was kindly provided by Cowper Sheetmetal & Engineering, of 11 Cowper St, Granville, NSW.

SPECIFICATION

A low-cost video data terminal designed to use any standard TV receiver or monitor as the display device. Interfaces to any microcomputer or other computer system via standard asynchronous 20mA input and output lines, at any one of nine standard data communication rates. Will also operate as a "TV typewriter". The keyboard circuitry has a 90-key ASCII encoder designed for use with low-cost surplus keyboards.

Main features are:

- Displays full 6-bit ASCII character set of 64 characters.
- Refresh memory capacity 1,024 characters, which may be organised as either 32 x 32-character lines or 16 x 64-character lines. Double-height characters possible in former mode.
- All critical timing derived from 2MHz crystal, including TV sync frequencies.
- Nine crystal-locked communication rates: 75,110,150,300,600, 1200, 2400, 4800, or 9600 baud.
- Full duplex or half duplex operation.
- Display roll-up and roll-down facility.
- Inbuilt audio alarm driver.
- Both video and modulated RF outputs.
- No critical setting-up.
- Low power consumption—approximately 5 watts.

ask if I thought we and our readers would be interested in a new low-cost video display module he had just developed, and was preparing to sell as a kit via his company E & M Electronics Pty Ltd.

We didn't talk for long before I realised that Ed's video module kit sounded as if it would make an excellent basis for our project terminal. And this was confirmed a couple of days later, when I visited Ed at his premises in Marrickville. His module proved to be a very impressive design, based on modern CMOS and low-power Schottky TTL integrated circuits, and with many worthwhile features.

For example the refresh memory has a capacity of 1,024 characters, many times that of earlier low-cost designs. Not only this, but Ed has given the module various display options, so that the memory may be displayed in a number of ways. You can have 32 lines of 32 single-height characters, for example, with 23 lines typically being visible on the screen at any one time. Or you can have the same number of lines and characters, but the characters displayed in double height mode—with about 11 visible at any one time. Or as a third option, you can have the memory displayed as 16 lines of 64 characters—with 15 lines visible at once.

Apart from this, one of the things which impressed me about the module was its lack of setting-up adjustments. Unlike many of the earlier designs, which had numerous preset adjustments to make (often using a CRO or a frequency counter), Ed's module has only two—both of which can be adjusted without instruments and when the module is finally operating.

Virtually all of the critical timing is derived from a single quartz crystal, including all of the video sync signals and the clock signals used to set the baud rates for communicating with the main computer system. This makes for very easy setting up, and for reliable operation as well.

The other features which Ed has provided on the module include both video and modulated RF outputs, optional roll-up and roll-down facilities for the display, a driver for a low-voltage sound generator (pulsed in response to the ASCII "bell" character code), and a range of nine standard communication rates from 75 to 9,600 baud, together with facilities for varying the serial data format as required.

In addition there is a switchable facility for half-duplex operation in place of the usual full-duplex mode, so that the display may be used to echo directly the characters from its associated input keyboard. This allows the terminal to be used away from computer systems as a "TV typewriter", for displaying text on TV receivers and video monitors.

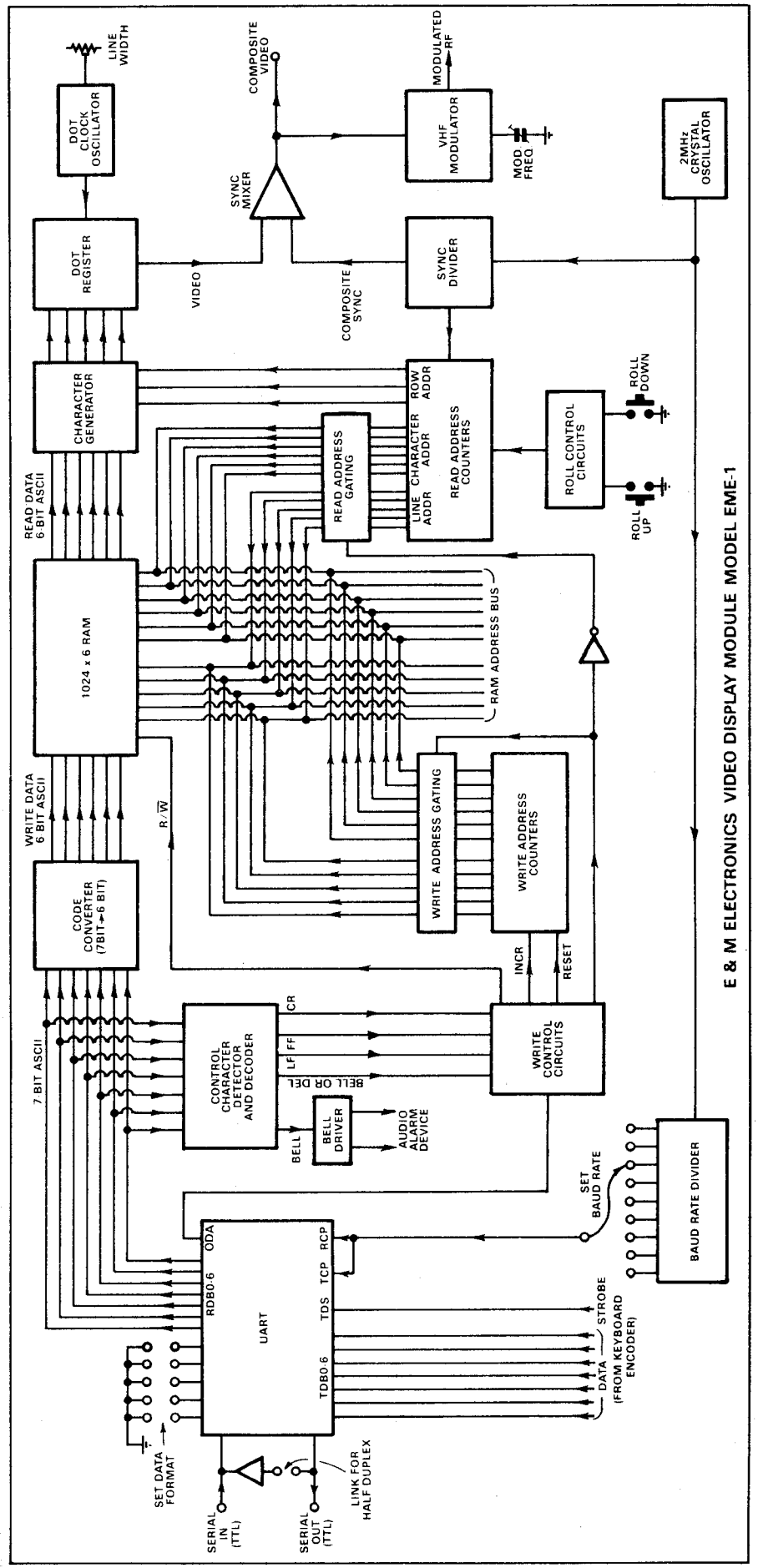


Fig. 1: Block diagram for the EME-1 video display module, heart of our terminal.

VIDEO TERMINAL

In short, Ed Monsour has produced a very flexible and practical video display module, one which is very suitable for use as the heart of a video terminal for microcomputer systems. All that is needed to produce a complete terminal is a keyboard with an encoder circuit, a simple interfacing circuit using opto-couplers to allow operation with any system having a standard 20mA serial input-output ports, and a power supply.

Thanks to the co-operation of Ed Monsour, we are able to present here the details of a complete terminal along these lines. We believe that this terminal will prove very suitable for use with any of the microcomputer systems currently available. and is likely to be compatible with many future systems.

And the good news is that you should be able to build the terminal for far less than any commercial video terminal. E & M Electronics are selling the EME-1 video display module kit for the very reasonable price of $199.00, including sales tax. So that you should be able to build the complete terminal for under $300, using one of the low-cost keyboards currently being advertised.

Incidentally for those who do not wish to assemble the module themselves, or are unable to do so, it is also available in completely assembled and tested form (at a higher price of course).

Both kits and assembled modules are available from E & M Electronics Pty Ltd, 136 Marrickville Road, Marrickville, NSW 2204.

For the remainder of the present article, we will describe the EME-1 video display module and the recommended way of assembling if from the kit supplied by E & M Electronics. Then next month we will describe a matching keyboard encoder, interfacing and power supply module we have developed to go with the video module, to turn it into a complete video terminal.

The full circuit diagram of the EME-1 module is supplied with the kit, but is too large to be reproduced properly here. However Fig.1 shows a fairly detailed block diagram, which should give you a good idea of the way it works.

From an operational point of view, the heart of the module is the refresh memory. This consists of a 6144-bit RAM, organised as 1024 6-bit words. The RAM uses low-cost static MOS chips, type 2102.

The remainder of the circuitry of the module is effectively divided into two distinct sections, one on each side of the RAM. On the "write" side of the RAM are the input circuits, which receive characters from the main computer system and perform the appropriate functions—writing the characters into the memory in the case of "printing" characters, or performing various housekeeping tasks in the case of non-printing or control characters.

On the "read" side of the RAM is the other section of the module circuitry. This section takes the characters stored in the RAM, and performs all of the tasks necessary to display them continuously on a normal video raster.

Let us look at the "read" section circuits first, as these involve concepts which may be a little less familiar. It is also these circuits which operate continuously, whereas the "write" section circuits only operate sporadically in response to characters arriving from the main computer system.

In essence, the job of the read circuits is to produce a continuous video signal, which when fed to a standard TV receiver or monitor will produce a steady display of the ASCII-encoded RAM memory contents in normal alphanumeric form.

The key circuit section involved in this task is the character generator. This is actually a high speed 2560-bit static ROM, organised as 64 groups of eight 5-bit words. Each group of eight 5-bit words corresponds to one of the 64 alphanumeric characters represented in the 6-bit subset of ASCII code.

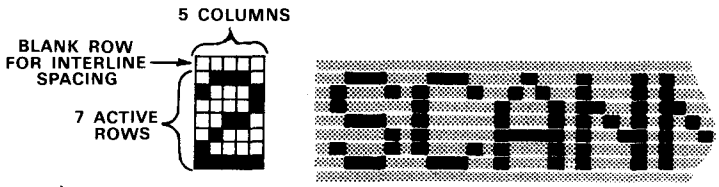


FIG. 2 : 5 x 7 MATRIX CHARACTER GENERATION

The eight 5-bit words corresponding to each character contain the information necessary to construct the character concerned in an 8 x 5 dot matrix pattern, as shown in the diagram on the left in Fig. 2. Each of the eight 5-bit words corresponds to one of the horizontal rows of the matrix, while the five bit positions in each word correspond to the five vertical columns of the matrix.

Actually the first 5-bit word of each character group is left blank (all bits zero), to provide vertical spacing between lines of characters in the final display. Only the remaining seven 5-bit words are used to construct the characters, so the characters are in fact formed from a 7 x 5 dot matrix.

The character generator ROM is organised so that 6-bit ASCII data words fed to it act as "character addresses", defining one particular group of eight 5-bit words. Then any of the eight 5-bit words containing the rows of dots required to make up the character may be arranged to appear at the ROM outputs, as required, by specifying a 3-bit "row address". Thus if the row address is 000, the blank spacing word is read out, while if the row address is 111 the word corresponding to the lowest row of the character will be read out.

A display with a line of characters like that shown on the right in Fig. 2 is produced by presenting the appropriate sequence of 6-bit ASCII code words repetitively to the character generator, for eight successive TV scanning lines. For the duration of each scanning line the character generator row address is kept static, but the address is incremented between lines.

As a result the character generator produces for each successive scanning line a sequence of the 5-bit words corresponding to the appropriate horizontal row of the line of characters. First a sequence of blanks is produced, then during the next line a sequence of the uppermost active rows for the line of characters. The third scanning line produces the sequence of "second active row" words, and so on down to the eighth scanning line which produces a sequence of words which correspond to the bottom row of the line of characters.

The sequence of 5-bit words produced during each line is turned into a serial video signal by the dot register, shown in Fig. 1 immediately to the right of the character generator. The dot register is basically a shift register, used here as a parallel-to-serial converter. The 5-bit row data words produced by the character generator are parallel loaded into the register, and then shifted out in serial fashion by feeding high speed clock pulses to the register.

The clock signals fed to the dot register are produced by the "dot clock oscillator". In the EME-1 the frequency of this oscillator is adjustable by means of a variable resistor (RV1), and this allows the width of the displayed lines of characters to be adjusted.

The serial output of the dot register is raw video information, and lacks the usual sync pulses. It is therefore fed to a mixer, where a composite vertical and horizontal sync signal is mixed with it to produce a normal composite video signal suitable for being fed to a standard TV monitor. The composite video signal is also applied to a VHF modulator circuit to produce modulated RF, suitable for feeding to the aerial terminals of a standard TV receiver.

The RF output frequency of the modulator may be adjusted by means of a trimmer capacitor, over a range which covers a number of the high-band Australian VHF TV channels. This allows the signal from the module to be placed on a blank channel, to prevent interference from local TV stations.

Note that the dot clock oscillator frequency adjustment and the VHF modulator frequency adjustment are the only two setting-up adjustments required on

the module, and neither requires instruments.

The horizontal and vertical sync frequencies produced by the module are derived via a 2MHz quartz crystal oscillator, via the sync divider circuits. As a result both sync signals are locked to within 0.2% of the standard TV frequencies of 15,625Hz and 50Hz.

In order that the character generator and its following circuits may produce the desired video display of characters, according to the description just given, the character codes stored in the RAM must be presented to the character generator in the appropriate sequences. This task is performed by the circuits represented in Fig. 1 by the block labelled "read address counters".

There are basically three counters in this section, one being a 6-bit counter which keeps track of the characters within the horizontal display lines. This is associated with a 5-bit counter which keeps track of the display lines themselves. Finally there is a 3-bit counter which provides the row address information for the character generator.

The outputs of the character and line address counters are fed to the address bus lines of the RAM via a set of gates. Normally these gates are enabled, so that the read address counters are able to cycle the RAM addresses over and over through the correct sequences for generating a continuous video display of the characters stored in the RAM.

There are actually two line address counters in the read address counter circuits, one an up-down counter and the other a normal up counter. It is the latter whose outputs are fed to the RAM address bus, and which increments once for each displayed line of characters. But at the start of each TV field, this counter is preset to a particular line address by parallel loading it from the up-down counter. Thus the count of the up-down counter determines on which line of the characters stored in the RAM the display begins.

By changing the count of the up-down counter, the display may thus be arranged to start on any desired line. This is the function of the roll control circuits, and the "roll up" and "roll down" push-buttons. One button increments the counter, the other decrements it.

Hopefully you can see from the foregoing that the RAM is normally under the control of the read circuits, and being used to produce the video display. The only time this state of affairs is interrupted is when a new character arrives from the computer system, causing the "write" circuits to spring into action.

As shown in Fig. 1, a UART is used to receive the asynchronous serial characters arriving from the main computer system, and convert them into parallel form. The transmitter section of the UART is made available for use by a keyboard encoder, to transmit asynchronously back to the system.

The UART receiver and transmitter clock signals are provided by a divider connected to the 2MHz crystal oscillator. The divider provides nine alternative output frequencies, corresponding to the standard asynchronous communication rates of 75, 110, 150, 300, 600, 1200, 2400, 4800 and 9600 baud.

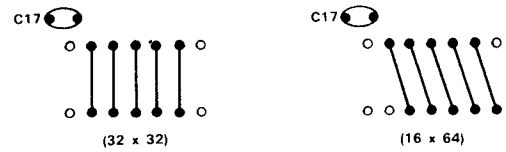
The data format used by the UART may be set in the usual way, by means

LINKS ON THE EME-1 VIDEO DISPLAY MODULE

A. DISPLAY FORMAT

Link	32 x 32 format	16 x 64 format
LK1	Join C-A	Join C-B
LK3	Join C-A	Join C-B
LK5	Join C-A	Join C-B
LK7	Leave open	Close
LK10	Join C-A	Join C-B

ADDRESS SWITCH LINKS:



B. CHARACTER SIZE

Link	Single height chars	Double height chars
LK4	Open	Close
LK8	Join C-A	Join C-B

(Note: double height characters available only in 32 x 32 format)

C. CHARACTER SPACING

To have display characters closely spaced, leave LK2 open.
To increase horizontal spacing between characters, close LK2.

D. LINE FEED OPTIONS

For instantaneous rack-up action on LF character (normal operation) join C-A of LK6 and leave LK9 open. Alternatively, display may be made to roll-up on LF by leaving LK6 open and closing LK9.

E. COMMUNICATION RATE

Speed	LK13	LK11	LK12
75 baud	Join L-A	Open	Open
110 baud	Join L-A	Close	Close
150 baud	Join L-B	Open	Open
300 baud	Join L-C	Open	Open
600 baud	Join L-D	Open	Open
1200 baud	Join L-E	Open	Open
2400 baud	Join L-F	Open	Open
4800 baud	Join L-G	Open	Open
9600 baud	Join L-H	Open	Open

F. DATA FORMAT

The links of LK14 set the usual serial data format:
A: No parity bit (NPB). Close to enable parity.
B: Number of stop bits (NSB). Close for 1, open for 1.5 or 2.
C,D: Number of data bits (NDB1 & 2). Both closed for 5 bits,
C only closed for 6 bits, D only closed for 7 bits, both open for 8 bits.
E: Parity odd/even (POE). Close for odd, open for even parity.
(Note: For normal teleprinter format used in most 110-baud interfaces, close links A and D only.)

G. FULL/HALF DUPLEX OPTION

LK15 controls this option. leave open for full duplex as required for interfacing with most computer systems employing automatic echoing of characters from keyboard. Close for half duplex, as required for systems not providing auto-echo, or for using the terminal as a "TV typewriter".

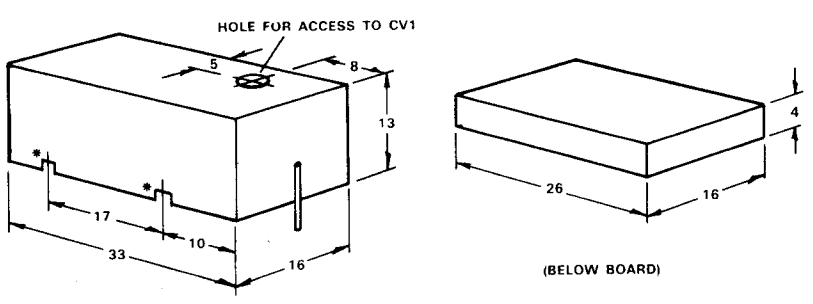
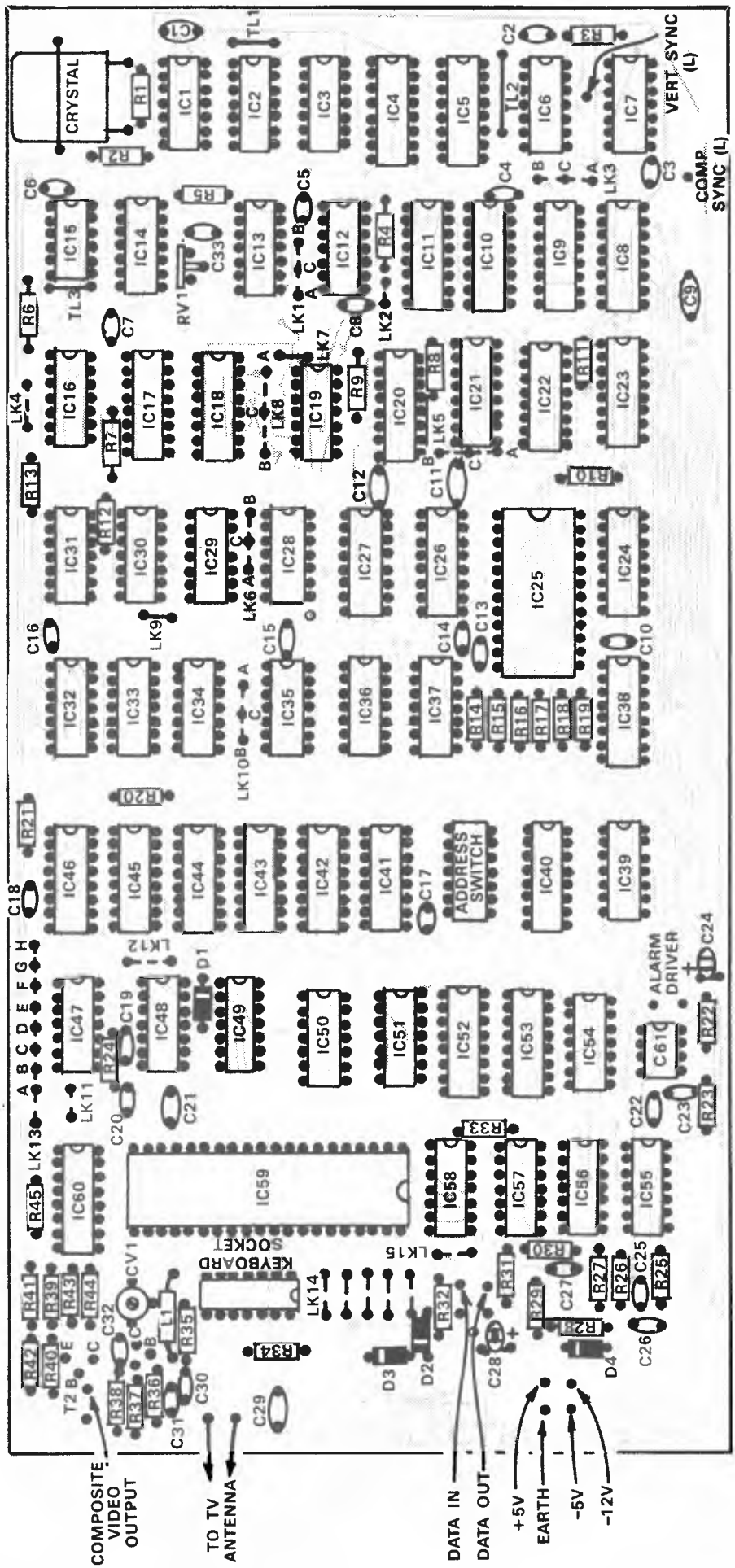


FIG. 4 : VHF MODULATOR SHIELDS



of links. A link may also be used to feed the output from the UART transmitter back to the receiver input, to provide half-duplex operation. Normally with this link open, the keyboard and UART transmitter operate quite independently of the UART receiver and the rest of the display module, giving full duplex operation.

The 7-bit ASCII data produced at the UART receiver outputs is fed to the RAM data inputs via a code converter circuit, whose function is to convert codes for the lower-case alphabetic characters into their equivalent upper-case character codes.

As well as being presented to the RAM via the code converter, the output data from the UART receiver is also fed to circuitry which detects and decodes the various non-printing control characters. The control characters which are detected and decoded are "bell", "line feed" (LF), "carriage return" (CR) and "form feed" (FF). The circuits also detect the "null" or "delete" character.

The decoded "bell" character is fed to an audio alarm driver circuit which is capable of pulsing one of the audio sound generators such as a Sonalert or Audiolarm. The remaining control characters and the delete character are fed to the write control circuits, where they are used to control various functions in response to the "output data available" (ODA) strobe signal from the UART.

For example if none of the control characters is decoded, nor a delete character, this is taken to imply that the received character must be a printing character. Accordingly the write control circuits increment the write address counter, and generate a control signal which disables the read address gates from the RAM address bus, and simultaneously enables the write address gates. This sets the RAM address bus to the next available RAM location, as specified by the write address counter. The write control circuits then send a WRITE (L) pulse to the RAM, causing the character to be written into that location.

If any of the control characters or the delete character is detected and decoded, the write control circuits inhibit this writing cycle. In the case of the delete and bell characters, nothing else is done. However, in the case of LF, CR and FF, the appropriate functions are performed instead. For an LF, the write line address counter is incremented to give a line feed; for a CR, the write character address counter is reset, to give the equivalent of a teleprinter carriage return; and for an FF both the write line address and write character address counters are reset, to cause the next printing character to be stored in the first

Fig. 3: Use the diagram at left as a guide to wire up the EME-1 display module. The parts are identified on the facing page.

RAM location—producing the equivalent of a form feed.

All of the circuitry of the EME-1 video display module is provided on a single PC board which measures 267 x 132mm (10.5 x 5.2 inches). The board is double-sided (etched conductors on both sides), and has plated-through holes. There are a total of 61 integrated circuits on the board, 14 of which are TTL (mostly low-power Schottky), one a 555 timer for the audio driver, and the remaining 46 are MOS devices.

The main reason for the relatively large number of devices is that designer Ed Monsour has used widely available and low cost SSI-MSI devices in preference to more costly and harder-to-obtain LSI types.

As the PCB pattern has many closely-spaced conductors, I suggest that you use a low-power soldering iron with a fine bit. I myself use a Mico 6V/10W "Mini" iron, with a bit having a tip diameter of about 1.2mm. This makes it somewhat easier to solder IC pins and components pigtails to the PCB pads without adding unintentional solder bridges.

I would also suggest that you use fine-gauge resin-cored solder, rather than the normal type, again to assist in avoiding bridges. For this type of work I myself use 0.7mm solder, which seems to be stocked by at least some suppliers.

To assist in wiring up the module from the E & M Electronics kit, we have prepared an overlay wiring diagram (Fig. 3). This shows all of the ICs and other components, listed numerically. The parts list gives the IC type numbers and component values, to enable each part to be identified. The wiring diagram also shows the various input and output points on the PCB, and the programming links. These are wired according to the information in the table.

In assembling the module, I suggest that you wire it up in the following sequence. First, fit the wire links, making sure that if bare tinned copper wire is used, the links are mounted proud of the PCB so that they don't short the etched pattern.

Note that there are three "test links" (TL1, 2 and 3) which must be fitted, as well as those described in the table.

Then fit all of the resistors, proceeding in numerical order according to the parts list and checking them off as you go.

Then fit the capacitors, following the same methodical routine. Don't forget to observe polarity with the two tantalum electrolytics, C24 and C28.

Now fit the four diodes D1-D4, making sure that each is orientated correctly. Also fit transistors T1 and T2, also making sure that they are correctly orientated. And fit inductor L1, mounting it slightly proud of the PCB.

Note that if you are sure that you will never be using the modulated RF output of the module, the VHF modulator components may be omitted. This includes R35-38 inclusive, C31 and C32, CV1, L1 and T1.

If you will be using the modulator, however, now would be the best time to fit the tin plate shields for it above and below the PCB. Details of the shields are given in the small diagram of Fig. 4.

Then fit the TTL integrated circuits, comprising ICs 7, 8, 9-15, 26-28, 57 and 58. Also the 555 device, IC61.

At this point you should pause to make sure that you are prepared for the MOS devices. Make sure that the soldering iron is earthed, and that the earth of the PCB is connected reliably to the barrel and bit of the iron via a cliplead. It is also a good idea to earth yourself, if there is any risk of you acquiring a static charge due to the action of plastic or rubber shoes on nylon carpet, etc.

Now fit the remaining MOS integrated circuits, working methodically from the parts list as before. When soldering in each one, it is a good idea to solder the supply pins first (usually pins 7 and 14); this makes sure the remaining pins are protected by the internal clamping diodes.

This completes the assembly of the video display module, and you should now be ready to add to it the remaining circuitry required to produce a complete video terminal. This will be described next month, in the second of these articles.

PARTS LIST FOR VIDEO DISPLAY BOARD

INTEGRATED CIRCUITS

IC1	4011	IC2	4024
IC3	4013	IC4,5	40163
IC6	4023	IC7	74LS00
IC8	DM8097	IC9-11	74LS161
IC12,13	74LS74	IC14	7400
IC15	74LS00	IC16	4001
IC17	4029	IC18	40163
IC19	4023	IC20	4029
IC21	40163	IC22	4029
IC23	40097	IC24	40163
IC25	2513-CM2140	IC26,27	74LS195
IC28	74LS08	IC29	4013
IC30	4011	IC31	4071
IC32,33	4013	IC34	4023
IC35	4001	IC36	4081
IC37	4024	IC38	40097
IC39	4024	IC40	40097
IC41-46	2102	IC47	4024
IC48	4023	IC49,50	4081
IC51	74C30	IC52,53	4028
IC54	4011	IC55	4071
IC56	4011	IC57	74LS04
IC58	74LS08	IC59	S1883
IC60	40163	IC61	NE555

TRANSISTORS

T1	2N706	T2	2N3643

DIODES

D1-D4	1N914

RESISTORS

R1	1M	R2	2.2k
R3	10 ohms	R4	10k
R5	2.2k	R6-10	10k
R11	2.2k	R12,13	10k
R14-19	2.2k	R20-22	10k
R23	33k	R24	10k
R25	1M	R26	100k
R27	10k	R28,29	22k
R30	2.7k	R31	10k
R32	470	R33	220
R34	470	R35	2.7k
R36,37	1k	R38	3.9k
R39	47ohms	R40	75 ohms
R41	2.2k	R42	1k
R43	3.9k	R44,45	22k
RV1	2.2k miniature variable		

CAPACITORS

C1-18	.01uF ceramic
C19	100pF ceramic
C20	.01uF ceramic
C21	.047uF
C22,23	.01uF ceramic
C24	6.8uF tantalum
C25,26	220pF ceramic
C27	680pF ceramic
C28	6.8uF tantalum
C29-31	.01uF ceramic
C32,33	10pF ceramic
CV1	3.5-13pF variable

MISCELLANEOUS

1	PC board, code EME-1 (See note below)
1	Inductor, L1
1	2MHz crystal

NOTE: All of the above components are sold as a complete kit for the EME-1 video display module by E & M Electronics Pty Ltd, 136 Marrickville Rd, Marrickville, NSW 2204. The proprietary PCB is not sold separately.

PHILIPS MONITOR ASSEMBLIES AVAILABLE

This is the video monitor assembly from a Philips model C3 30-cm monochrome TV receiver. Although it does not include a sync separator, it will interface to our video data terminal quite easily. Cost of the assembly is $125, and stocks are available from Cema (Distributors) Pty Ltd, who have offices in Sydney and Melbourne.

Video data terminal for microcomputers—2

In this second article presenting our easy to build video terminal design for microcomputer systems, the author describes the keyboard encoder, interfacing and power supply module. He also describes the final assembly of the terminal, and discusses optional switching facilities.

by JAMIESON ROWE

As we noted in the first of these articles, the EME-1 video display module mates with a normal TV receiver or video monitor to provide virtually all of the circuitry for the "receive" side of a video data terminal. To form a complete data terminal, all that is required is a suitable keyboard and encoder, some simple interfacing to mate with 20mA current-loop communication lines, and a suitable power supply. These facilities are described in this article.

The full circuit for the remaining sections of the terminal is shown in Fig. 5, and as you can see they are quite straightforward. The keyboard encoder is based on an LSI MOS encoding device, the National Semiconductor MM5740AAF. This is a scanning-type encoder, capable of dealing with up to 90 key switches arranged in a X-Y matrix. It provides 7-bit ASCII encoding, with automatic code changing for shift and control modes.

Basically the MM5740 consists of two ring counters, a ROM containing the ASCII codes for 90 different characters, and control logic. One ring counter has 9 bits, and is fed with clock pulses; its outputs are fed via drivers to energise each of the keyboard matrix "X" lines, so that the lines are energised one after the other on a cyclic basis.

The other ring counter has 10 bits, and its outputs are used to enable sensing gates connected to the 10 "Y" lines in the keyboard matrix. This counter is also fed with clock pulses, whose frequency is such that all 10 possible switch positions on each "X" line are sensed while that line is energised. Hence the MM5740 continuously scans the full 90 possible switch matrix locations.

This scanning is performed at a high rate, so that when a key is depressed, its closure is detected within a very short time. The MM5740 then uses the outputs of the two counters as an address in its internal ROM, to look up the corresponding ASCII code.

Actually there are four separate sections of the internal ROM, one containing the codes for normal key weighting and the others the codes for shift mode, control mode and shifted control mode. Which ROM section is used to produce the code depends upon whether or not the "shift" and/or "control" keys are currently pressed.

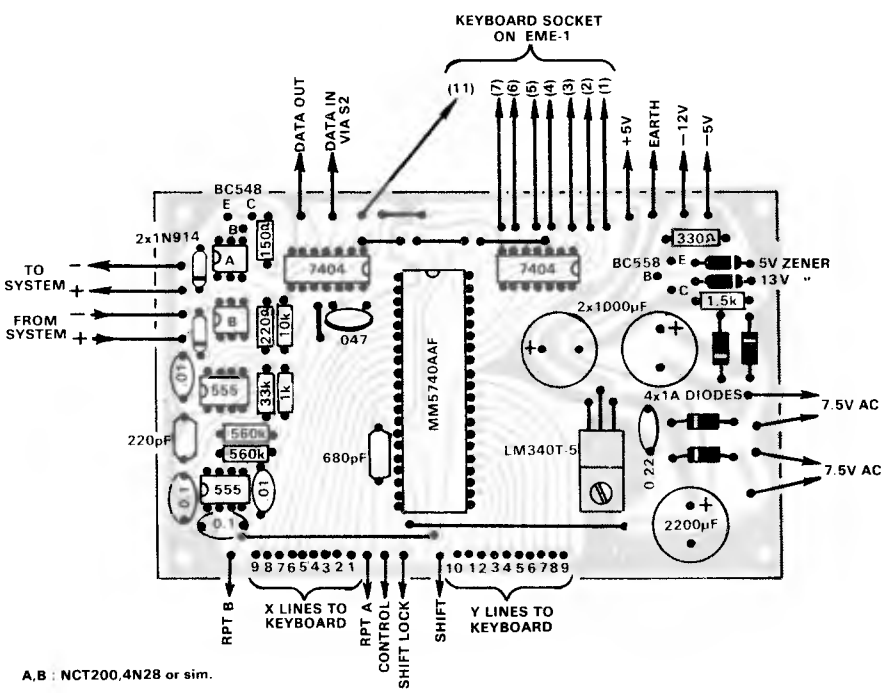
The character code read from the ROM is fed into a set of data latches, and made available to external circuitry via terminals B1-B9. At the same time a "key pressed" or data strobe pulse is generated at the DS output, to indicate that a new code has been generated.

Bits B1-B7 are the actual ASCII output code outputs. Bit 8 is an internally generated even parity bit, for use if this is needed, while B9 is a selective repeat bit for systems which require one. Neither B8 or B9 is required for a simple data terminal of the type we are concerned with here. If parity is required, this may be generated by the UART on the display board.

A nominal 100kHz clock signal is required by the MM5740AAF for its keyboard scanning, and this is provided by a simple clock oscillator using a low-cost 555 timer IC. A second 555 oscillator running at approximately 10Hz is used to provide for character repeating, via the "repeat" key. This is a convenient facility for horizontal tabulation using spaces, etc.

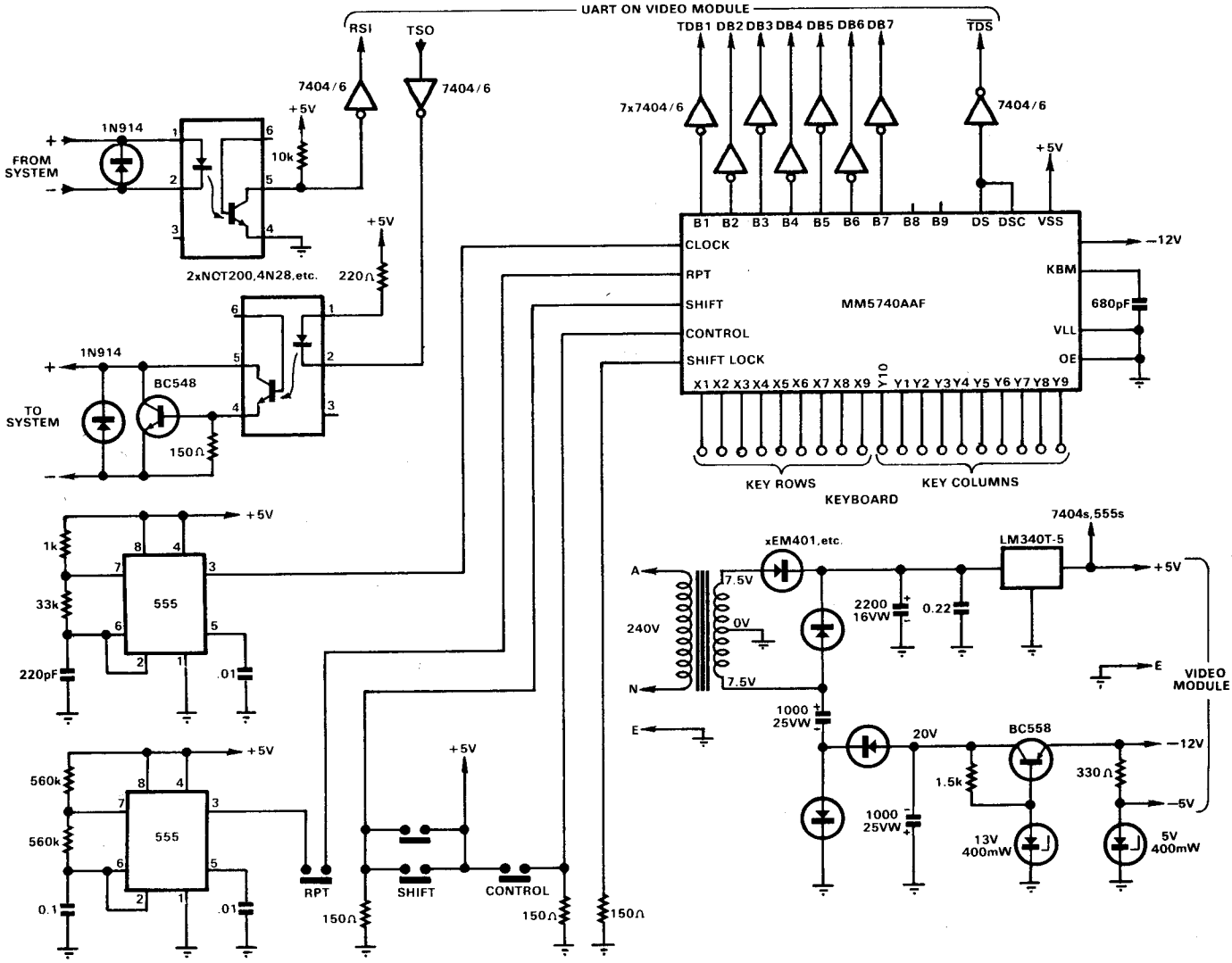
The MM5740AAF encoded data outputs B1-B7 use the negative logic convention, whereas the transmitter data inputs of the UART on the EME-1 board require positive-true data. Hence a set of inverters are used to invert the data, using 7404 inverter elements. A further 7404 inverter is also used to invert the strobe output of the encoder, for although the encoder in this case generates a positive-true pulse, the UART requires a negative-true pulse at its TDS (transmitter data strobe) input!

The 20mA current-loop interfacing is very simple, and is based on two low-cost opto-coupler devices such as the NCT200, 4N28 or similar. These provide a means of coupling the TTL-level serial input and output ports of the UART device to and from normal 20mA current-loop lines, with full isolation. This allows the data terminal to communicate with virtually any computer system,



A,B : NCT200,4N28 or sim.

Fig. 6 at left shows the component positions and links on the encoder PCB.



KEYBOARD MODULE FOR E & M VIDEO TERMINAL

Fig. 5: The circuit for the rest of the terminal, apart from the keyboard.

regardless of the DC potentials of the 20mA loops.

The 20mA line coming from the computer system is connected to the LED of one opto-coupler, whose phototransistor is coupled to the RSI (receiver serial input) pin of the UART via another 7404 inverter. The inverter gives the correct logic polarity, so that the RSI input of the UART is held at the correct positive logic level for the 20mA "mark" condition of the line.

The TSO (transmitter serial output) pin of the UART is connected to the LED of the other opto-coupler via a further 7404 inverter, so that when the TSO output is at the positive logic "mark" level, the LED is conducting. The phototransistor of the coupler is then connected in Darlington configuration with a BC548 or similar transistor across the line to the computer system, so that in the "mark" state the two are conducting and completing the loop.

Note that 1N914 or similar diodes are connected across the two 20mA line connections, to protect the opto-couplers from damage in the event of reverse polarity.

The power supply requirements of the complete video terminal are quite modest. The EME-1 display module requires +5V at around 300mA, −5V at around 15mA, and −12V at around 15mA also. The keyboard encoder and its 555 clock generators and inverters also require +5V at a few tens of milliamps, and the encoder also requires −12V at a few milliamps.

These requirements are provided by the simple power supply circuitry shown. A single tranformer is used, with a centre-tapped 15V secondary rated at 1 amp. A full-wave rectifier using the full winding and two 1A silicon diodes with a 2200uF reservoir capacitor is then used to feed an LM340-T three terminal IC regulator, to produce the +5V supply.

A half-wave voltage doubler rectifier using two further 1A silicon diodes and two 1000uF capacitors is used to derive the two negative supply voltages. The output from the rectifier is fed through a simple series-pass transistor regulator using a 13V zener diode in the base of a BC558 or similar transistor, to produce a nominal −12V supply. A simple shunt regulator circuit fed from this supply is then used to produce the −5V supply.

Virtually all of the circuitry shown in Fig. 5 is mounted on a small PC board measuring a modest 132 x 80mm. The PCB pattern is coded 77ut2, and etched boards made to this pattern should be available shortly from board suppliers. They should be low in cost as the PCB is single-sided, small in size and does not involve a large number of holes.

Wiring of the PCB should be fairly straightforward, as we have prepared a wiring diagram with the etched pattern "ghosted" in light grey as if the board were held up to a light (Fig. 6).

When you wire up the board, I suggest that you fit the wire links first, then the resistors and capacitors, and finally the semiconductors. As the MM5740 encoder IC is fairly expensive, it might be wise to use a 40-pin socket rather than solder it directly into the board. But use a high grade socket, or you may get contact troubles. The remaining ICs and tran-

sistors are best soldered directly into the board.

Watch the polarity of the diodes and electrolytic capacitors, also the orientation of the ICs. Note that the LM340-T regulator IC should have a heatsink flag of at least 40mm square bolted to its tab, to keep its temperature down. I bent one up from a scrap of 16 gauge aluminium sheet, so that it had a vertical surface and a "foot" clamped between the IC tab and the PCB by a 3mm bolt and nut.

Note that there are three sets of interconnections between the encoder PCB and the video module described last month. The encoder output data and the strobe output connect to the keyboard socket, with the pin numbers shown in Fig.6. The four power supply connections go to the group of four pads on the video module adjacent to D4, down in the lower right-hand corner of Fig. 3 given last month. Similarly two serial data connections go to the pins between R31 and R32 on the video module, with the "data in" wire going via a switch if you want switchable "TV typewriter" operation. More about this later.

We have designed the encoder PCB so that the keyboard matrix "row" and "column" lines are simply brought out to pads along the side. This makes it possible to use virtually any suitable keyboard, although you have to wire up the keyswitches to the encoder PCB using conventional wiring. The alternative would have been to design the PCB so that a particular keyboard would simply "drop in", with the etched pattern performing all the connections. But this would have produced a large and costly PCB, as well as making it difficult for readers to use other keyboards.

For the prototype I used one of the keyboard assemblies currently being offered by Dick Smith Electronics Pty Ltd, as this firm was kind enough to provide a sample. The keyboard is a sturdy plastic moulding, with gold-plated contacts in the switches. It has most of the keytops required for an ASCII-encoded format, although there is no "carriage return" keytop. However, there are a couple of blank keytops on switches which are otherwise spare, and one of these can be used as the carriage return key. The keytops are not all in the usual positions when you get the keyboard, but they can easily be swapped around as desired. Do this before you wire up the switches to the encoder, though, or you'll get rather confused!

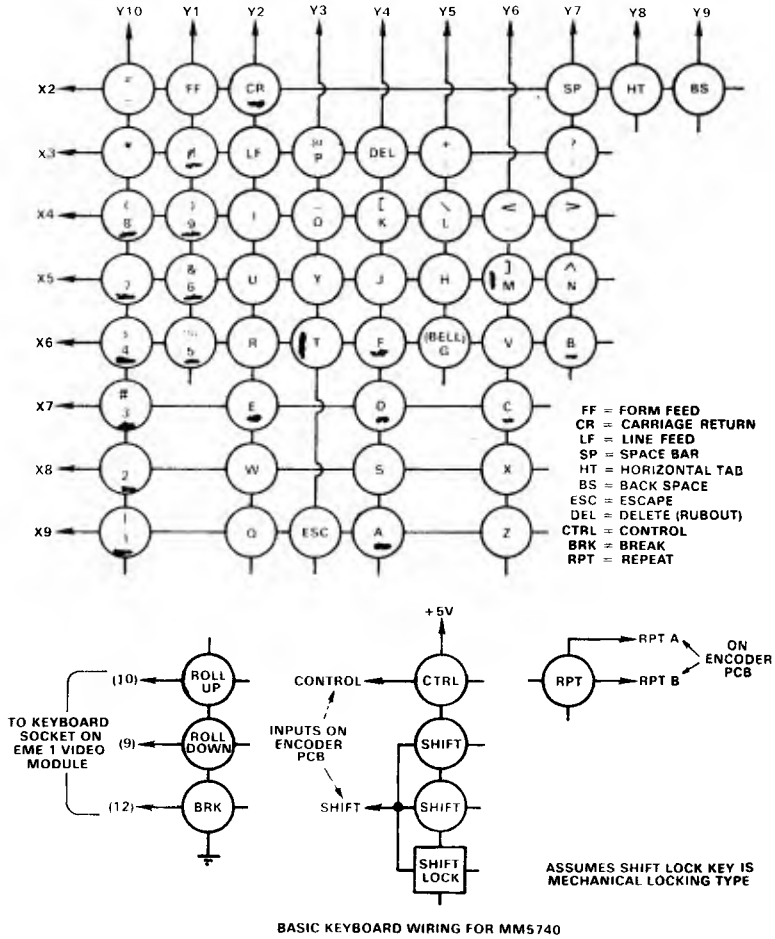


Fig. 7: How to wire up the keyboard switches to the encoder.

Other keyboards could of course be used, including some of those from surplus keypunch equipment. But check that the keytops are suitable for ASCII encoding—some keyboards have the numerals as "upper case" symbols on the keytops, or keytops with strange combinations of characters which just "won't work" with the MM5740 encoder.

Even the keyboard from DSE has a few keytops which are not suitable for the MM5740. Two of these may be used for the "roll up" and "roll down" keys for the EME-1 module, but the others may simply be ignored.

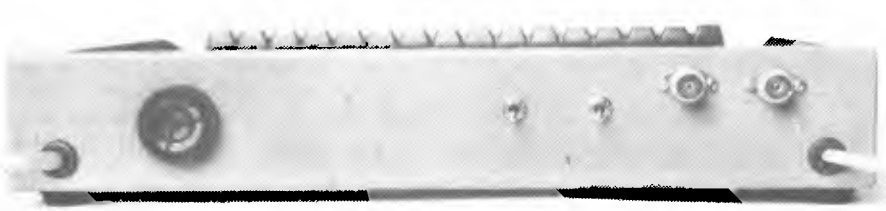
Fig. 7 shows the way the various keyswitches are wired up to the encoder board. Most of the keys are wired in the main encoder scanning matrix, formed by the column lines Y1-10 and the row lines X2-X9. Note that row line X1 is not used for a normal typewriter-style keyboard as used here.

The remaining switches connect to either the control inputs of the encoder PCB, or the keyboard socket of the video display module. Those that connect to the encoder board are the control key, the two shift keys (wired in parallel), the shift lock key, and the repeat key, while those that connect to the video module keyboard socket are the roll up, roll down and break keys. The last of these also performs UART resetting, if this is ever needed.

Of the keys that connect to control inputs on the encoder PCB, note that all but one have the +5V rail as their common connection. The exception is the "repeat" key, which has one side going to the 10Hz oscillator output and the other going to the encoder IC.

Note that the circuit of Fig. 7 assumes that the shift lock key on the keyboard is of the mechanically locking type. This is the case with the key on the DSE keyboard, and also on some others.

If you have a keyboard with a non-locking "shift lock" key, this can still be used because the MM5740 device has an internal shift-mode latch. The connection to it is via pin 20 of the IC, and is brought out to a pad on the encoder board as you



This view of the rear of the terminal shows the audio alarm, two small toggle switches for terminal-TV typewriter switching, and the output connectors.

You can see from this view of the inside of the terminal how easy it is to assemble, thanks to the two PCB modules. The second "R" key visible on the right was used on the prototype as the "CR" key.

can see in Fig. 6. The way to connect up a non-locking key to this pad is shown in Fig. 8. Note that the circuit provides for a 12V 40mA indicator lamp, to show when the MM5740 is in the shift mode. The lamp driver transistor may be a 2N2222, BC548 or other general-purpose NPN silicon type.

Using the two PCB modules we have described, you should be able to build up our video data terminal either as such or as a "TV typewriter". Some readers may elect to build it up as one or the other, knowing that they are never likely to want it to perform the other function. Similarly they may connect up the links for a particular serial communications rate, say 110 bauds, and leave it at that.

However as some readers may wish to have the device switchable in terms of function and data rate, we have indicated in Fig. 9 how this may be done.

As you can see, we have shown three switches. Switch S1 is used to control the hardware echo facility, so that its two positions are marked "full duplex" and "half duplex". Switch S2 is then used to make or break the line from the opto-coupler to the UART serial input, so that it becomes a "normal-TV typewriter" switch.

Note that for normal full-duplex operation, both switches should be in the positions shown. For half-duplex data communications, S1 alone should be changed, while for "TV typewriter" operation both switches must be changed to the other positions.

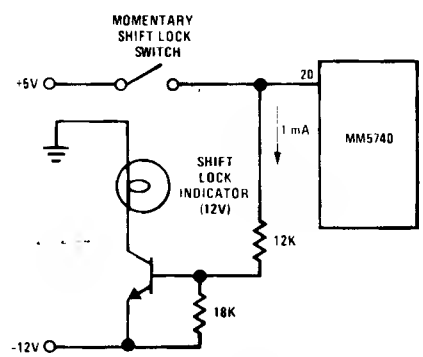
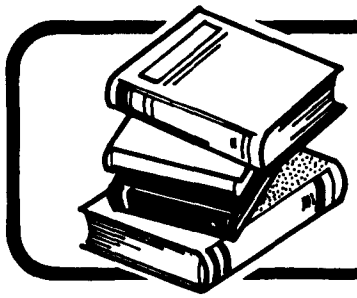


Fig. 8: How to wire up a shift-lock key which is not mechanically latching.

The third switch shown, S3, illustrates how you can switch data rates if desired. Although only a two-position switch is shown switching between 110 and 300 bauds, there is no reason why you could not have a switch with further positions if you wish. However, if one of the speeds you want is 110 bauds, as shown, the switch must be a three-pole type in order to be able to control the links LK11 and LK12.

We have shown switching for only the 110 and 300 baud rates because these are the two speeds used on most of the smaller microcomputer systems.

Also shown in Fig. 9 are the connections for the audio alarm device, to allow the terminal to respond to the "bell" character. Any alarm device designed to operate from a nominal 6V supply and drawing up to 200mA may be used. We used a Bell Audiolarm from C & K Electronics, but a Sonalert could also be



Books & Literature

Microcomputers

AN INTRODUCTION TO MICROCOMPUTERS, Volume 1. Published by Adam Osborne and Associates, Berkeley, California, 1976. Soft covers, 134 x 205mm, many diagrams. Price $12.50.

AN INTRODUCTION TO MICROCOMPUTERS, Volume 2. Published by Adam Osborne and Associates, Berkeley, California, 1976. Soft covers, 134 x 205mm, many diagrams. Price $15.00.

8080 PROGRAMMING FOR LOGIC DESIGN, Published by Adam Osborne and Associates, Berkeley, California, 1976. Soft covers, 134 x 205mm, many diagrams.

Back in October last year we reviewed the second printing of the first edition of the Adam Osborne "Introduction to Microcomputers", and noted that it was shortly going to be released as two separate volumes. Not only has this happened, but the book has already begun to spawn further offspring!

What has apparently happened is that as planned, the "basic concepts" part of the first edition was expanded and split off to become the new Volume 1. That left the remainder of the first edition, the material dealing with specific microprocessor chips and their systems, to form Volume 2. But by the time the publishers came to produce the second volume, two things became apparent. One was that there were a number of new devices, which should be included; the other thing was that each device really needed more detailed treatment than previously.

The most obvious result of this is that Volume 2 has now grown in size to more than double that of the complete first edition. Even so, the publishers explain that they have really only been able to provide what they regard as a sufficiently detailed and thorough coverage of four devices: the 8080A, the MC6800, the Z80 and the MCS6500.

In addition, because they feel that users need more help with the detail of programming and applying the various devices, the publishers have announced that they will be issuing a whole new series of books on specific device use. The first of these is already available, for the 8080; the second dealing with the 6800 is due to appear shortly.

Briefly, volume 1 is a basic introduction to microprocessors and microcomputers. It assumes very little background knowledge, apart from a basic understanding of electronics. It is concise and thorough, and if read carefully gives a sound grasp of virtually all the basic concepts of microcomputers.

Volume 2 then leads you into the real world of specific devices. There are 19 different devices or device families treated: the TMS1000, the F8, SC/MP, the 8080A, the Z80, the MC6800, the MCS6500, the PPS-8, the 2650, COSMAC, the EA9002, the IM6100, the SMS300, PACE, the CP1600, the TMS9900, the mN601 and 9440, the 2900 and 6700 series, and the MC10800 series.

Broadly speaking each device is first placed in general context, then its internal logic and addressing modes are described. Device pins and signals are described, together with interfacing and interrupt techniques. The instruction set is given, together with a benchmark program to illustrate various strengths and weaknesses.

In summary, the two volumes of "An Introduction to Microcomputers" are now even better than the first edition as an introduction to the subject and a reference. They would be almost essential reading for anyone planning to work with microprocessors and systems.

The first of the new programming books deals appropriately with the 8080. In brief, it is a detailed and down-to-earth guide for the logic designer, to help in making the change to microcomputer-based logic design.

Like the first two books it is concise, thorough, and well planned. As such it should be of great value to anyone seeking to become proficient in the design of 8080-based systems.

The review copies came from Dick Smith Electronics Pty Ltd, who advise that they currently have stocks of the first two, with catalog numbers B2340 and B2342 respectively. (J.R.)

VIDEO TERMINAL

used—or even a small bell or buzzer.

As you can see from the photographs, the prototype terminal was assembled in a low-profile case made from 20G mild steel sheet. The case measures 360mm wide by 400mm deep by 60mm high, with a sloping front cutout to allow the keyboard to mount in a convenient operating position. The case is designed so that a small portable TV receiver or monitor may be placed on top, to form a complete video terminal.

The prototype case was kindly supplied by Cowper Sheetmetal and Engineering, of 11 Cowper Street, Granville, NSW 2142. It was finished in chocolate lacquer, and looks very attractive.

Assembly of the circuitry inside the case is quite straightforward. The two PC

REMAINING PARTS:

1 PC board, 132 x 82mm, coded 77ut2
1 Keyboard encoder IC, MM5740AAF (National Semiconductor)
1 5V/1A three-terminal regulator, LM340T-5
2 555 time ICs
2 NCT200, 4N28 or similar opto-couplers
2 7404 hex inverter ICs
2 1N914, 1N4148 or similar diodes
4 1A silicon diodes, OA626 or similar
1 13V 400mW zener diode
1 5V 400mW zener diode
1 BC548 or similar NPN transistor
1 BC558 or similar PNP transistor

RESISTORS Half watt, 5%:

4 x 150ohms, 1 x 220ohms, 1 x 330ohms, 1 x 1k, 1 x 1.5k, 1 x 10k, 1 x 33k, 2 x 560k.

CAPACITORS

1 220pF polystyrene or NPO ceramic
1 680pF polystyrene or ceramic
2 .01uF greencap (polycarbonate)
1 .047uF greencap
2 0.1uF greencap
1 0.22uF greencap
2 1000uF 25VW PC-type electrolytic
1 2200uF 25VW PC-type electrolytic

MISCELLANEOUS

1 Case, 400 x 360 x 60mm with sloping-front (see text).
1 15V 1A stepdown transformer
1 ASCII-type keyboard assembly (see text)
1 Audio alarm, Sonalert or Bell Audiolarm
2 Co-axial sockets
Miniature toggle switches, as required
Mains cord and plug, section of B-B connector strip, cord clamp; length of 4-way cable for system cable, with suitable connector; rubber feet, assembly screws, etc.

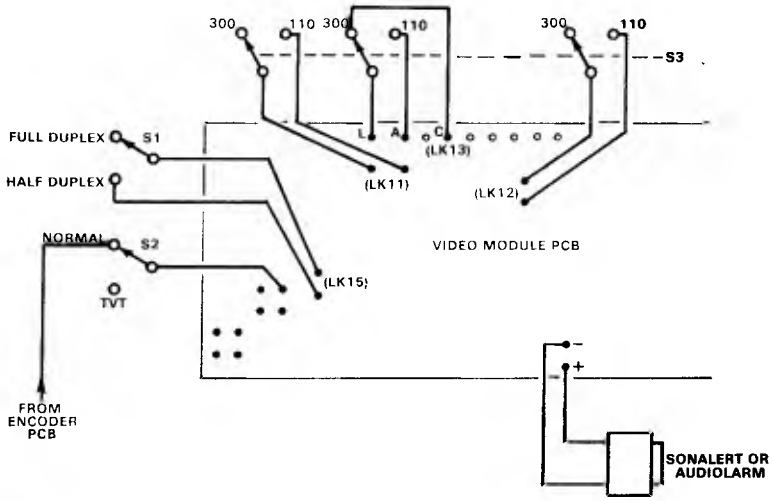


Fig. 9: Details of the wiring for the duplex and TVT switches S1 and S2, together with optional baud rate switching via S3.

boards are simply mounted on spacers or "Richco" plastic moulded PCB mounts, with the video module about 35mm from the rear of the case, and the smaller encoder module in front of it alongside the power transformer. Incidentally we used the Ferguson PF3597, a low-profile type which is now given the type number PL15/20VA. However, the case height has been designed to allow most of the available 15V/1A transformers to be used—including the A & R 2155A and the DSE 2155.

The mains cord enters through a grometted hole in the case rear, is clamped and the active and neutral leads terminated in a B-B connector which also terminates the transformer primary leads. The mains cord earth conductor is looped around and soldered to a lug which is bolted to the metal case.

Also mounted on the rear of the case are the coaxial sockets for the video and modulated RF outputs (for the display), the optional function and communication rate switches, and the audio alarm device. The communication cable to the main computer system also leaves the case at the rear, via another grometted hole.

The only remaining item is the keyboard assembly, which is mounted in the front of the case so that its keys protrude through the cutout in the removable top-front. If you use the DSE keyboard, as we did, the easiest way to mount it in position is to bend up a couple of shallow brackets out of strips of aluminium sheet, with a shape best described as a shallow inverted-U with unequal legs. The legs can be bent as required to obtain the correct height and slope for the keyboard in the case, and then mounting holes drilled in the feet and the bottom of the case, to screw it in position.

We also mounted rubber feet on the underside of the case, to prevent scratching the surface on which the terminal is placed. The case top is attached using small self-tapping screws.

When you have completed the assembly, it would be a good idea to check through all of the interconnections before applying power. Then when power is applied, the terminal should operate straight away.

There are only two adjustments, as noted in the first article, and both of these may be done without instruments. The first is to set the frequency of the VHF modulator, to allow clear display on a TV receiver tuned to a vacant high-band channel. This is done by simply tuning the receiver to a suitable channel, and then adjusting the trimmer capacitor CV1 on the video module until the image appears and is best displayed.

Don't forget to remove the normal antenna while doing this, however, and don't forget to remove it subsequently whenever you are using the terminal with a normal TV receiver. Otherwise you'll get interference from TV transmissions, and the neighbours may also find themselves getting interference from your terminal!

If you do this adjustment when the terminal has just been powered up, you'll find that the display will show a lot of "garbage"—jumbled alphanumeric characters and symbols, in random order. This is merely the random turn-on contents of the display refresh RAMs, and is quite normal. To clear the display, simply type in a series of line feed characters. Then type the "form feed" key, which homes the display circuitry so that the next character to arrive will appear in the bottom left-hand corner of the screen.

Now type a full line of characters—32 if you have set up the display module for 32 x 32 format, or 64 if you have set it up for the 16 x 64 format. This will allow you to make the remaining setup adjustment, for line width. All you do is adjust RV1, on the display board, until the lines are of a suitable width on the screen.

In normal operation, the terminal operates in very similar fashion to an ASR-33 teleprinter. The only real difference is that the "print-out" is displayed on the TV screen, and that you are able to roll the display up and down as desired using the two special keys.

Of course there is no facility for dumping programs onto paper tape, and subsequently re-loading them. However we hope to provide a suitable low-cost alternative for this also, in the near future.

Alternative keyboard with encoder:

Readers planning to build a video terminal may be interested in this keyboard assembly available from the Microswitch Division of Honeywell Pty Ltd, at 863 Burke Street, Waterloo NSW 2017. It is complete with a MOS keyboard encoder providing ASCII output, and features Hall-effect keyswitches which offer bounce-free, high reliability operation. Designated the 54SW5-3, the keyboard has a mechanically latching shift-lock key, but no CR or LF keys. A CR keytop is available, however, and the correct coding can apparently be generated. Price for the keyboard is $110 plus 15% sales tax. The CR keytop is 20c extra.

Cassette tape interface for microcomputers

Here is a simple and effective way of storing and recalling digital data strings on an unmodified audio tape recorder. The unit is intended to connect between a computer and a terminal unit using normal asynchronous serial data transmission implemented with 20mA current loops, and operate at a variety of transmission rates.

by DAVID EDWARDS

There are many possible ways of storing digital data and computer programs that will provide ready access, flexibility and reliability. The choices available to a home computer enthusiast are rather limited, however. The "fortunate few" interact with their computer via a Teletype model ASR-33 teleprinter, and can thus store and recall programs and data on punched paper tape.

Other enthusiasts (usually with less financial resources), use a "glass" terminal, such as that recently described by Jim Rowe. With such a terminal device, however, it is not possible to store and recall programs or data.

For those with access to a normal audio tape recorder (a cassette unit is ideal), help is at hand, as such machines can be used for this purpose.

Most of the small mini-computers and microprocessor evaluation kits currently available are intended to interface with a teleprinter or similar device using standard 20mA serial current loops. Data is transmitted and received in standard asynchronous serial form. Each bit lasts for a little over nine milliseconds, with the stop bit transmitted continuously during breaks in transmission.

Data in this form cannot be recorded directly on a normal audio tape recorder, but must be suitably conditioned. Normal practice is to use "frequency-shift keying" or FSK, recording digital one signals as a 2400Hz tone, and digital zero signals as 1200Hz tones. This is consistent with the standard resulting from the "Kansas City" symposium, sponsored in the USA by BYTE Magazine.

The unit described in this article is designed to connect between the computer and the terminal unit, without affecting the operation of either. The output data stream from the computer is passed directly to the terminal as normal, and at the same time is processed and made available at the recorder input.

To record a program or data listing, all that is required is that the program or data be "dumped" into the terminal, while the recorder is recording.

A switch is provided to enable the computer input to be switched between the terminal output and the conditioned recorder output. Normally, the serial information provided by the terminal is passed directly to the computer. When the switch is placed in the replay position, the data stream from the recorder is conditioned, and then passed to the computer. The computer software or firmware would normally have been instructed to enter a "load" phase prior to this being done.

In accordance with normal practice, the input and output connections to the computer are isolated with optocouplers. There are only three adjustments to be made during construction, none of which are critical. Two are simply input and output level controls for the tape recorder, which are adjusted to suit the particular recorder in use.

The recording input signal for the recorder can be adjusted from 0 to 4 volts peak to peak, which should cope with all commonly available recorders. The sensitivity of the replay input is 400mV peak to peak, or about 140mV RMS, which again should cope with most recorders.

The remaining adjustment is that of the interface's system clock. This is mainly to set the frequencies recorded on tape at exactly 2400Hz and 1200Hz, so that tapes made on one system will be compatible with those made on another system. This setting is not at all critical if the unit is only ever used to replay its own recordings, however, as the inherent design of the unit ensures that within limits it will always be compatible with itself.

This is because the two frequencies recorded on tape are separated on replay by a digital filter which is in effect tuned by the system clock frequency. Since the tones on the tapes are derived from the clock also, the filter will always be matched to them. This system will cope with recorder speed variations of up to 30% without error, although naturally enough, the data stream will be stretched or compressed in time.

Recapping, then, the purpose of the clock adjustment is only to make tapes made on one system compatible with those made on another, and also compatible with the Kansas City standard.

We have arranged that the data switching between the two tones recorded on tape occurs synchronously, so that no harmonics are generated which could possibly interfere with the operation of the recorder, by interaction with the bias oscillator. This does mean that the data pulses are altered in length by a small amount, but this does not prove serious in practice.

By now you may be wondering what the cost of all these features is going to be. Well, rest assured. Even though there are twelve integrated circuits (ICs), they are all low cost, readily available linear and digital TTL parts. Only a single 5V power supply rail is required, and this is provided by a three terminal regulator.

The component parts are assembled on a single printed circuit board (PCB), measuring 142 x 132mm. Apart from a case, two switches and the input and output connectors, all that is required is a transformer. We will leave these details up to the individual constructor, as we feel that for this sort of project, this is the most flexible approach.

We are indebted to Ed Monsour of E & M Electronics for much helpful advice with the initial design of the circuit. In fact, our system was developed from an initial design of his.

Turning now to the circuit diagram, we can discuss the circuit operation in some detail. The clock is formed by a 555 timer, connected in the astable mode. T1, a BC337 transistor is used to buffer the output, to maintain a reliably low "0" logic level when two TTL loads are driven. The clock frequency is a nominal 24kHz.

A 7490 decade counter is used to divide the clock frequency by ten, and provides the 2400Hz signal required by

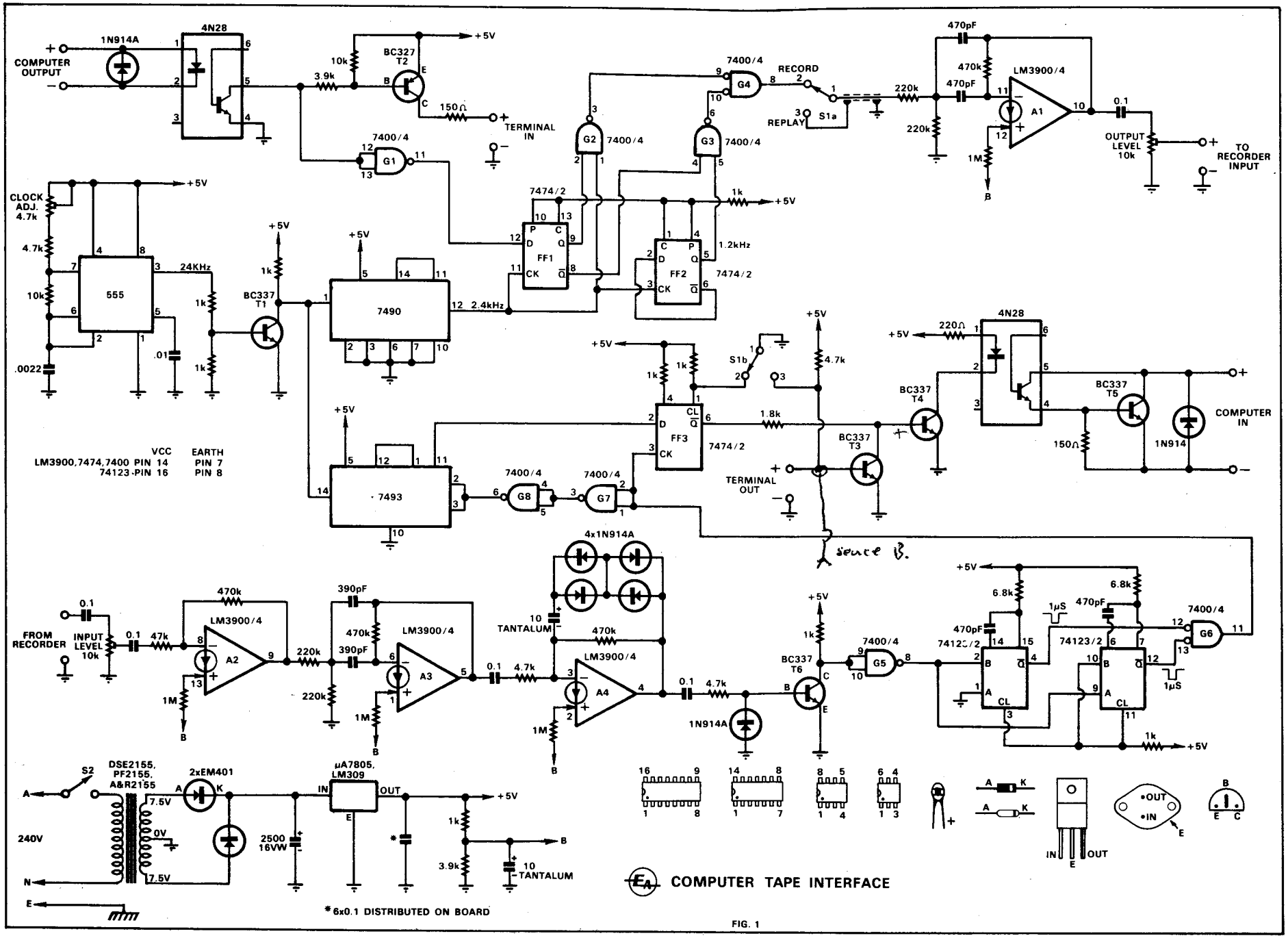

COMPUTER OUTPUT
1N914A
4N28
BC327 T2
10k
3.9k
150Ω
TERMINAL IN
7400/4
G1
G2 7400/4
G3 7400/4
G4
RECORD
REPLAY
S1a
470pF
470k
220k
LM3900/4
A1
1M
0.1
OUTPUT LEVEL 10k
TO RECORDER INPUT
CLOCK ADJ. 4.7k
4.7k
555
24KHz
.0022
.01
BC337 T1
7490
2.4kHz
7474/2
FF1
FF2
1.2kHz
1k
+5V
S1b
4.7k
1.8k
FF3
7493
G7
G8
BC337 T3
BC337 T4
BC337 T5
220Ω
150Ω
1N914
COMPUTER IN
TERMINAL OUT
VCC EARTH
LM3900,7474,7400 PIN 14 PIN 7
74123 PIN 16 PIN 8
4x1N914A
10 TANTALUM
390pF
A2
A3
A4
47k
BC337 T6
G5
G6
6.8k
1µS
74123/2
FROM RECORDER
INPUT LEVEL 10k
S2
DSE2155, PF2155, A&R2155
2xEM401
240V
7.5V
0V
2500 16VW
µA7805, LM309
IN
OUT
3.9k
*6x0.1 DISTRIBUTED ON BOARD
COMPUTER TAPE INTERFACE

FIG. 1

the recorder. One half of a 7474 D type flip-flop is then used to generate the 1200Hz signal also required. The second half of the 7474 is used, along with a 7400 quad gate, to switch synchronously either of these tones as dictated by the data input.

The computer output is isolated by an optocoupler and then passed to the terminal by T2, connected as a switched 20mA current source. The optocoupler output is also buffered and inverted by a gate G1, and fed into the data input of flip-flop FF1.

This flip-flop is clocked by the 2400Hz signal, so that the information at the data input can only be passed to the output at the positive going edges of the clock input signals. This means that the input data pulses may be stretched by a maximum of 0.833mS (2 2400Hz cycles), compared to the data pulse width of 9.091mS (assuming 110 bauds).

The Q and Q-bar outputs of the flip-flop are used to control gates G2 and G3, which pass either the 2400Hz or 1200Hz signals, depending on the input data. The gate outputs are summed in gate G4, and then passed to switch S1a.

When this switch is in the record position, the coded data is passed through a bandpass filter to the recorder input. The filter is formed from one quarter of an LM3900 quad op-amp, and has a centre frequency of 1500Hz, a gain of 1 and a Q of 1. This allows it to pass either tone equally well, while attenuating out of band signals, notably the higher harmonics of the input squarewaves.

The output from the filter is AC coupled to a level control preset, and then passed to the recorder input. In the replay mode, the input to the filter is grounded, to prevent possible feedback problems with the recorder, and to prevent the TTL signals from coupling into the remaining amplifiers.

During replay, data at the Q-bar output of FF3 controls T4, which drives an optocoupler. The optocoupler output is buffered by T5, which forms the switch required by the computer input circuitry. The terminal output (a normally closed switch of some sort), also controls T4 by way of T3, and allows keyboard data to pass through to the computer.

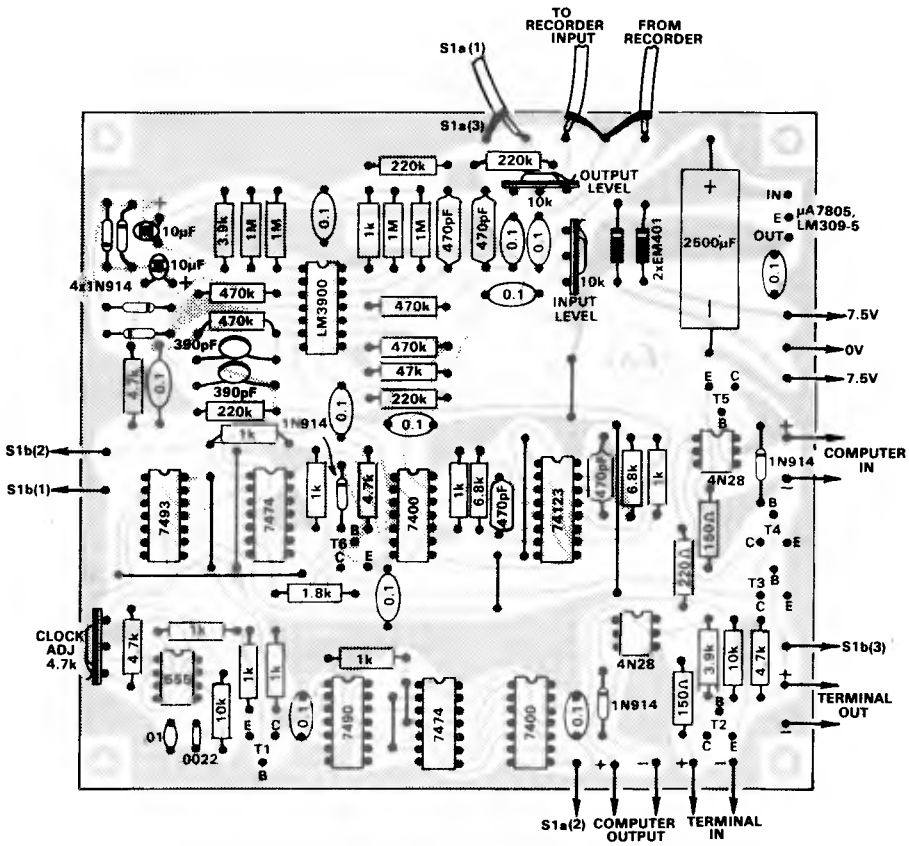
Switch S1b is used to prevent these two sources from attempting to send data to the computer at the same time. In the record position, FF3 is held reset, and the Q-bar output remains high, allowing T3 to control T4. In the replay position, the base emitter junction of T3 is shorted, holding it off. This allows the Q-bar output of FF3, which is controlled by the signal from the tape, to control T4.

During replay, the tones from the recorder first pass through a level preset, and are then amplified in A2 by 20dB (a gain of 10). They then pass through a second bandpass filter (A3), to eliminate mains hum and tape hiss. This filter has a gain of 1, a Q of 1 and a centre frequency of 1800Hz. It is nominally identical to the filter used for recording purposes.

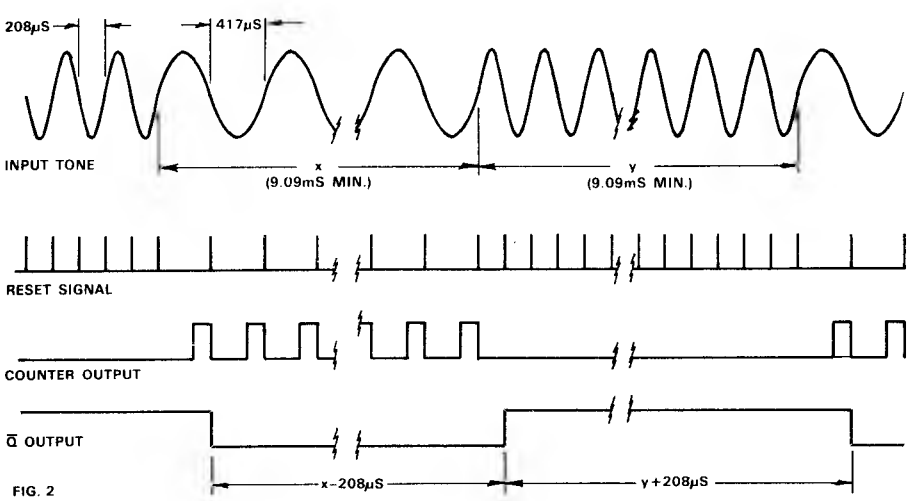
Ideal filters for this purpose would have a centre frequency of about 1700Hz (the square root of 1200 x 2400). This would ensure that the upper and lower cutoff frequencies would be symmetrical with respect to the 1200Hz and 2400Hz tones. With preferred value components, however, it is difficult to achieve this centre frequency.

The best we were able to achieve was 1800Hz and 1500Hz centre frequencies, and fortunately it turns out that if we use both of these, one in each filter, the errors tend to cancel out. This means that after the tones pass through both filters, their amplitudes will be roughly equal.

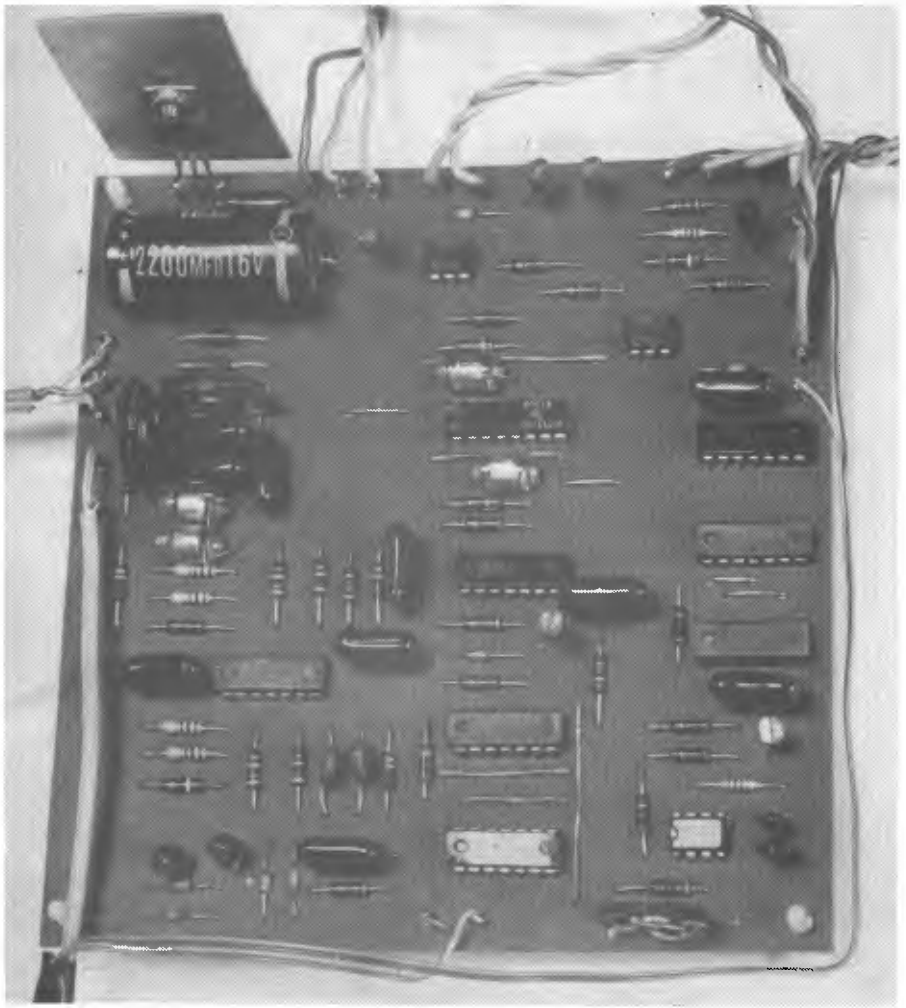
If more accurate equalisation is desired, it is possible to use 390pF capacitors in both filters, and to pad them with 22pF capacitors. This will give nominal centre frequencies of 1700Hz, as required. Normal component tolerance variations will introduce some errors, however.



The component layout shows the PC board from the component side. Use PC stakes to facilitate connections to power supplies and supporting equipment.



This timing diagram shows the relationship between the input tone, the reset signal, the counter output, and the output latch of flip-flop FF3.



All components are accommodated on a single printed circuit board.

PARTS LIST

SEMICONDUCTORS
1 555 timer IC
2 4N28, NCT200 or similar optocouplers
2 7400 quad gates
2 7474 dual D type flip-flops
1 7490 decade counter
1 7493 4 bit binary counter
1 74123 dual monostable
1 LM3900 quad op-amp
1 LM309, uA7805 or similar 5V 1A regulator
5 BC337 NPN silicon switching transistors
1 BC327 PNP silicon switching transistor
7 1N914A silicon diodes
2 EM401 silicon diodes

RESISTORS (all ¼W)
2 150 ohm, 1 220 ohm, 8 1k, 1 1.8k, 2 3.9k, 4 4.7k, 2 6.8k, 2 10k, 1 47k, 4 220k, 4 470k, 4 1M
1 4.7k trimpot (0.2in lead spacing)
2 10k trimpots (0.2in lead spacing)

CAPACITORS
2 390pF polystyrene
4 470pF polystyrene
1 0.0022uF polyester
1 0.01uF polyester
11 0.1uF polyester
2 10uF tantalum electrolytics
1 2500uF 16VW electrolytic, pigtail type

MISCELLANEOUS
1 printed circuit board, coded 77cc4, 142 x 132mm
1 transformer, 240V to 15VCT @ 1A, A&R 2155, PF 2155, DSE 2155 or similar
1 SPST switch
1 DPDT switch
Solder, tinned copper wire, hookup wire, shielded cable, scrap aluminium (for heatsink), PCB pins
NOTE: Resistor wattage ratings and capacitor voltage ratings are those used for our prototype. Components with high ratings may generally be used provided they are physically compatible.

The filtered replay tones are then clipped by the remaining LM3900 amplifier, which has four diodes connected in the feedback loop. The output from this amplifier is a 2.4V peak-to-peak squarewave with relatively steep transitions between levels. This waveform is squared up even more by a transistor switch (T6), and then inverted and buffered by a gate, G5.

The output from the gate is then passed to two monostables formed from a 74123 device. One monostable triggers on positive going edges, and the other on negative going edges. Each monostable is set to give an output pulse width of 1us. The negative going pulses from the monostables are summed in a gate, G6.

The output of this gate is a train of positive going pulses, synchronised to the zero crossings of the input tones. This signal forms one of the two inputs to the digital filter. The second input is simply the same signal, slightly delayed in time by the propagation delay of two further gates, G7 and G8.

The digital filter itself is formed by a 7493 binary counter and a D type flip-flop, FF3. The input of the counter is driven continuously by the 24kHz system clock signal, while the counter is reset at every zero crossing of the input tones by the delayed 1us pulses.

The clock frequency has been chosen so that when the input tone has a frequency of 2400Hz, with a half period of 208.3uS, five clock cycles each 41.7uS long will be counted between each reset. Under these conditions, the most significant bit of the counter will always remain low.

When the input tone has a frequency of 1200Hz, the half period will be 416.7uS, and ten 41.7uS clock pulses will be counted. This means that the most significant bit of the counter will go high for some time at the end of each half period of the recorded tone, as shown in Fig. 2.

Thus we can tell the frequency of the incoming replay tone by looking at the most significant bit of the counter: if this bit stays low, the input is 2400Hz, and if it goes high for some time, the input tone is 1200Hz. The higher frequency tone can be 30% lower in frequency, and the lower tone 30% higher in frequency, before the most significant bit will give an erroneous indication.

The most significant bit of the counter is connected to the D input of the flip-flop, which is clocked by the non-delayed zero crossing pulses. This means that the flip-flop samples the counter output just before it is reset, and stores this output till the counter is next reset.

This has the effect of shortening the "low" output pulse by 208uS, and lengthening the "high" output pulse by 208uS. Since one output pulse is stretched, and one is shortened, the errors tend to cancel. In any case, 208uS is short compared to the shortest pulse normally found in the system (9.09mS long), and we have not found this error to cause any problems with either computers or terminals.

The power supply is simply a rectifier-

Microcomputer News & Products

Shugart "minifloppy"

Floppy discs are now well established as an efficient and reliable storage medium for computer programs and data. Unfortunately the cost of conventional floppy disc drives and controllers has tended to price them out of the microcomputer area.

Happily this situation should now improve, thanks to the SA400 Minifloppy disc drive released in the US late last year by Shugart Associates. Virtually a scaled-down version of the well proven Shugart SA800 drive, the new unit takes discs only 133mm in diameter (5.25in), stores up to 109.4 kilobytes, and sells for only 2/3 the price of its larger brother: around $460 plus tax.

The performance is still very high, largely because the head and other key parts are identical with those used in the larger devices. Data transfer rate is 125 kilobytes/second, with a recording density of 2600 BPI maximum. Average access time is 550 milliseconds. Soft error rate is $1/10^9$, hard error rate is $1/10^{11}$, and seek error rate $1/10^6$.

Track spacing is 48 per inch, and there are 35 tracks. Both hard and soft sectored discs are available.

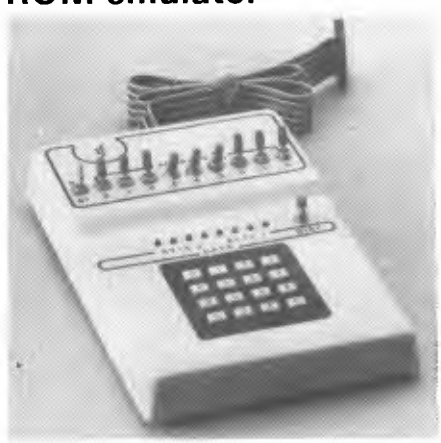
Size of the SA400 is 83 x 146 x 203mm, with a mass of 1.36kg. Total power dissipation is only 15W. No AC power is required, as the drive motor, track stepping motor and read-write electronics all operate from +5v and +12V DC.

Shugart also makes a matching controller module, the SA4400 Ministreaker. This can control up to three SA400 drives, but costs more than a basic SA400 drive.

Stocks of both the SA400 and SA4400 are now available from the Australian agent for Shugart, Warburton Franki Pty Ltd of 199 Parramatta Rd, Auburn NSW 2144.

The SA400 minifloppy drive would seem ideal for use by serious microcomputer hobbyists and other small users. However, in view of the relatively high cost of the SA4400 controller, here at EA we are currently working on the design of a low cost controller to allow the SA400 to be interfaced to small systems. We hope to present the design in the near future.

ROM emulator



The Sunrise Electronics KPRAM is a ROM and PROM emulator, to facilitate rapid and efficient development of dedicated microcomputer systems. It contains RAM, a hexadecimal data entry keyboard, toggle switches for addressing, and a set of LEDs for data checking. A program can thus be fed into the RAM or modified at any time, while at the same time connected into a system emulating a ROM.

Further information from Warsash Pty Ltd, PO Box 217, Double Bay, NSW 2028.

Computer tape interface

filter capacitor combination driving a three terminal 5V 1A regulator. Six 0.1uF bypass capacitors are distributed about the circuit, to eliminate possible switching spikes.

Construction of the PCB assembly should be relatively easy. All ICs can be soldered directly to the board, using a minimum of heat and solder. There are nine wire links, which should be insulated. All resistors should be 5% ½ or ¼W types. The three trimpots required should have 0.2" pin spacings. Care is required with the orientation of the electrolytic capacitors, diodes, transistors and integrated circuits. Use the PCB overlay diagram as a guide to placement of components.

External connections to the board are best made with circuit board pins. The connections to the recorder sockets, and to S1a1 and S1a3 are best made with shielded cable, as shown on the overlay diagram. The remaining connections can be made with colour coded hookup wire.

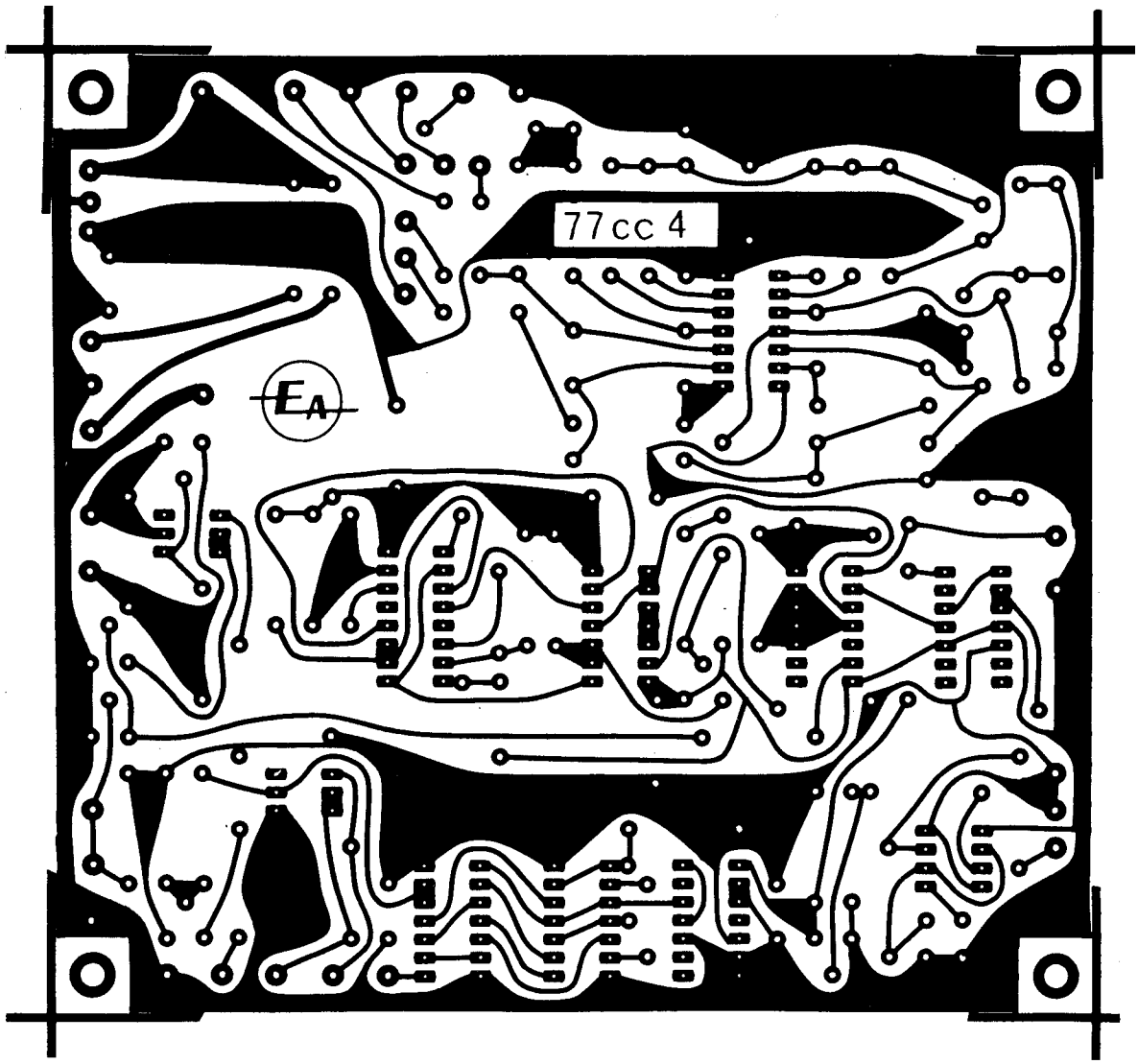
Do not forget that a heatsink will be required for the regulator IC. This can be either a scrap piece of aluminium bent up into an "L" shape, mounted next to the board, or the case itself. If the leads to the regulator are more than about 100mm in length, it may be necessary to wire a 0.1uF capacitor directly across the input and common pins, to ensure stability.

Once construction is completed, the board can be tested and adjusted. Monitor the +5V rail on initial switch on, and switch off immediately if it does not go to 5V ±0.25V. Assuming all is well, the clock can be adjusted. If you have access to a equency meter, simply monitor the coll tor of the buffer transistor, and adj t for 24kHz.

If you do not have access to a frequency meter, connect a computer and terminal, and while keying the terminal, adjust the clock frequency so that the 1200Hz and 2400Hz waveforms at the recorder input are equal in amplitude. To achieve even better alignment, connect the recorder input terminal to the recorder output terminal, and adjust the clock for equal amplitude signals at pin 5 of the LM3900.

This adjustment procedure will not set the clock at exactly 24kHz, but it will put it in the ball park, and will provide optimum performance if used alone. If you are attempting to read tapes made on another system, with possibly a slightly different clock frequency, then the clock can be adjusted during playback for minimum errors.

The preset level controls are adjusted so that the appropriate levels are supplied to and obtained from the re-



Here is a full-size reproduction on the PC pattern.

corder. During playback, it is better to have the input amplifier clipping rather than have too small a signal, so that sudden drops in level will not cause errors.

In use, the interface unit is simply interposed between the computer and the terminal. With the switch in the record position, the computer and terminal will interact as normal, and any data and programs can be recorded simply by switching on the recorder (in its record mode, of course).

To replay programs back into the computer, send the appropriate load commands from the terminal to the computer, then switch to replay, and finally put the recorder in the replay mode. During replay, the terminal keyboard is disabled, but the terminal will still display the computer output, which will usually allow indication that the load has been completed.

After the load is completed, switch the interface back to "record" to resume normal operations.

At this point a few words regarding the type of recorder which can be used are in order. In general, any machine which is capable of recording and replaying 1200Hz and 2400Hz sinewaves without causing appreciable distortion and level changes will be adequate. Machines which use an AC bias system, rather than a DC bias system, are to be preferred, as these usually give more consistent results.

While reel to reel machines can be used, we feel that a cassette recorder is better, mainly because of the ease of tape handling. While a tape counter is not vital, it does give a convenient means of indexing and finding particular programs or data groups.

Units with automatic level controls can be used, as well as manual models, although the former is a little more convenient to use. Be warned that some portable recorders do not like working into a high impedance, such as that provided by the interface unit, and it may be necessary to provide a nominal load (e.g. 22 ohms) for the output stage.

The unit is also capable of working with cassette decks, as used in hi-fi systems. Simply adjust the level presets to suit the lower signal levels commonly found in such machines.

Finally, some discussion of accuracy is warranted. We have dumped and loaded 200 bytes of memory a good many times with recording and replaying done on different machines without error. This suggests that substantially error free operation should be obtainable with most types of recorders.

There may be occasional errors due to tape dropouts, momentary loss of tape-head contact and so on, however. To minimise these, only good quality tapes should be used. We recommend that a tape be replayed immediately after recording, to check for errors. If the tape gives errors, replay it again as a second check, as errors will sometimes occur in playback.

If the tape still refuses to replay without errors, make another attempt at recording. We suggest that you keep a hard copy of all programs so that if the worst comes to the worst, you can always feed them in by hand again. With a good recording, performance during replay largely depends on the setting of the output level.

Using cassette tape for program and data storage:

Why not try the NRZ recording technique?

An alternative approach to the use of cassette tapes for storing computer programs and data is to use the "non-return-to-zero" or NRZ method favoured by professional computer designers. If you are prepared to forego the use of a standard audio recorder and work "from scratch" with a basic tape transport mechanism, it isn't as hard as you might think.

by IAN CROSER*

Most of the cassette tape interface designs so far described for microcomputers have used the audio frequency-shift keying (FSK) approach. This is capable of quite good results at low data rates, but involves a number of problems if reliable operation is required at higher rates. It is also fairly critical in terms of recorder performance and tape quality.

In comparison, the non-return-to-zero or NRZ recording technique used in commercial digital recording offers a number of advantages. Probably the main advantage is that it is relatively tolerant of tape deck and head performance, and also of tape quality. This is because it uses saturation-level recording, where the tape is always saturated in one magnetic direction or the other.

Another advantage of NRZ recording is that it is more efficient, allowing you to record higher bit densities and hence store more information on a given length of tape. At the same time it is essentially somewhat simpler than the FSK approach, requiring neither recording bias or the need to erase. As the recorded levels are relatively high there is also very little problem with hum and noise.

To be sure, you can't use a normal audio tape recorder for NRZ recording. It is necessary to start with a bare tape mechanism, and provide the rather different type of electronics required for NRZ recording and playback. But this turns out to be rather simpler than you'd think.

The circuit shown has been developed for use with one of the "Vortex" cassette tape deck mechanisms which have been available for some time. It does not require the deck to be modified significantly, using the record/play head as supplied. The erase head is not used, but may be left in position.

The circuit is designed for normal asynchronous serial data input and output, as used by most small microcomputer systems. It will operate quite reliably at data rates up to 1000 bits per second at the normal cassette speed of 1-⅞in per second.

Input and output levels are compatible with the RS232 voltage interfacing standard, as used by many systems.

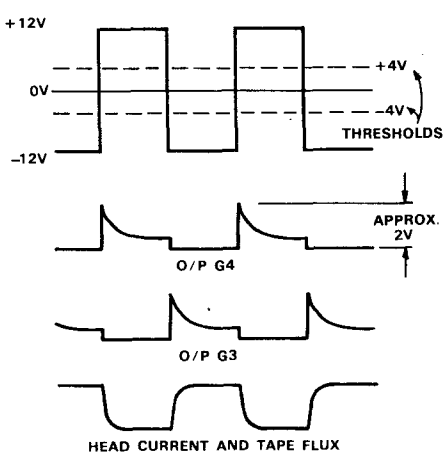


FIG. 1 RECORD MODE

However, it would be relatively simple to adapt the system for 20mA current-loop interfacing if desired. As shown the input impedance is approximately 10k, with thresholds at ±4V; output levels are ±11V with current limiting at 40mA.

As you can see from the circuit diagram, the complete interface uses only two ICs apart from the power supply. Analog functions are performed by three amplifiers of an LM324 quad op-amp device, while the digital functions are performed by the elements of a 7407 hex inverter.

Amplifier A1 is the input threshold detector, used to produce digital levels from the incoming data signals. Its threshold levels are set at ±4V approximately by the ratio of R1 and R2.

Resistors R15, R16 and diode D1 clamp the output of A1 to provide TTL logic levels for G1. This is one element of the 7407 hex open-collector inverter, used as a buffer.

Inverters G2, G3 and G4 provide complementary drive for the tape head, in record mode. Resistors R6 and R7 are used to set the head current at ±5mA.

Inverters G5 and G6 are used to disable the recording drive circuitry in the playback mode. When the R/P switch is set to the P position, resistor R3 takes the inputs of G5 and G6 to the logic high level, so that the outputs of both inverters fall to the low level.

This forces the outputs of both G3 and G4 to rise to their high-impedance logic high level, ensuring that data input is prevented from reaching the head. Note that a 7407 device has been chosen to reduce leakage currents in G3 and G4 in replay mode.

As you can see, this recording circuit simply converts the incoming digital data into bipolar current levels through the recording head. This is shown in Fig. 1. Note that although the inductance of the recording head produces "spikes" in the voltage waveforms fed to it from G3 and G4, the head current and hence the magnetic flux recorded on the tape are sub-

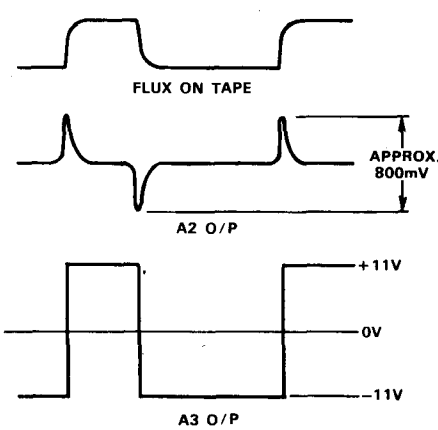
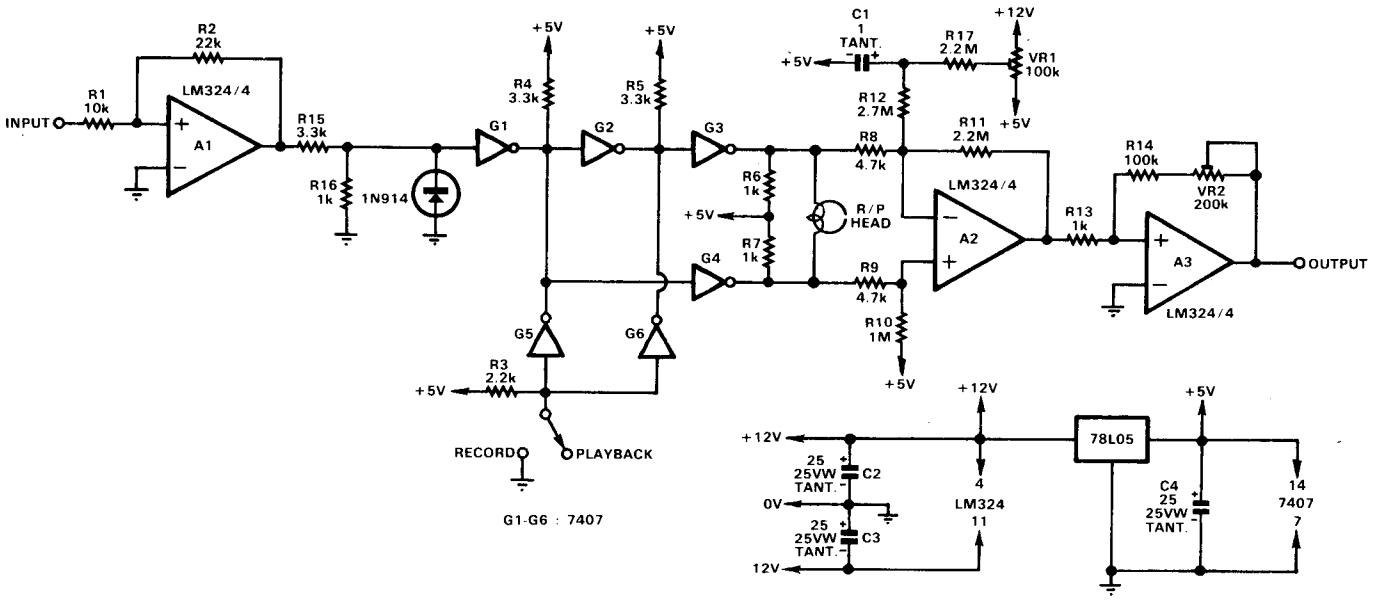


FIG. 2 PLAYBACK MODE

*c/o P.O., Stirling, S.A. 5152



The complete circuit for the author's NRZ digital recording unit, which apart from the power supply uses only two ICs—an LM324 quad op-amp, and a 7407 hex inverter. No erase head is required, as the recordings are at saturation level.

stantially the same as the incoming data.

A logic 1 in the data is recorded as a saturation flux level in one direction, and a logic 0 as a similar flux level in the opposite direction.

If a 1 is followed by another 1, or a 0 by another 0, the recorded flux does not change in either level or direction. Hence the description of this type of recording as non-return-to-zero.

For replay, A2 is used as a high gain differential-input amplifier to boost the head signal to detectable threshold levels. R8, R9, R10 and R11 set the differential voltage gain at approximately 500 while VR1 is used to adjust the output offset voltage to zero.

The offset adjustment circuit is filtered by C1, which is tied back to the +5V rail because this is the amplifier's input reference.

Amplifier A3 is used as the replay threshold detector. Its hysteresis levels are adjusted by VR2 for reliable decoding of the data. Its action is shown by the waveforms in Fig. 2.

As you can see, the amplified replay signal from the head consists basically of pulses corresponding to the polarity reversals of the recorded flux. Due to the hysteresis of A3, its output only changes levels when these pulses cross the two threshold levels. Hence the NRZ recording is converted back into a normal digital pulse train.

The thresholds of A3 are normally set at 50% of the peak pulse levels from A2, for reliable operation.

The circuit as shown requires two power supply voltages, +12V and −12V. A 78L05 or similar low-power three-terminal 5V regulator is used to derive the +5V supply for the 7407 from the +12V rail. The supply voltages are not particularly critical, although the +12V rail should be within ±1V and well filtered.

Typical current drain is less than 60mA from the +12V rail and less than 50mA from the −12V rail, but the exact drain depends upon output loading.

The tape mechanism was not modified except for the record switch, which was set to close contacts when the record button is pushed. As noted earlier, the erase head is not used as a new recording automatically erases anything previously on the tape.

The type of cassette used is not critical, although "wide dynamic range" tape will probably give the best results. Also as the speed tolerance is entirely a function of the data format, it would probably be desirable to use C30 cassettes rather than longer-play types with a thinner and less stable base.

The second channel of the tape head may be either used separately, or wired in series or parallel with the first, for more reliable operation.

I have not described the physical construction of the tape interface, as this is best left to the individual constructor. As few parts are involved, the complete circuit could be built underneath the Vortex mechanism if desired.

Simple power supply for microprocessor systems

Many of the microprocessor development systems and evaluation kits now becoming available require dual power supplies. Rather than tie up variable bench supplies, it is usually more convenient to use a dual fixed-voltage supply which can be "dedicated" to the job. The simple supply described here has been designed with this in mind.

by JAMIESON ROWE

When we first started to work with microprocessor evaluation kits and development systems a few months ago, the easy way to get them "up and running" in the shortest time was to power them from one or more variable bench supplies. However it soon became apparent that this was not the ideal long-term approach.

For one thing, bench supplies have a habit of disappearing when one's back is turned! Like most development labs, ours never seems to have enough bench supplies, and as a result any supply which is not actually in use on one bench tends to be "borrowed" and taken to another.

In itself, perhaps, this is no more than an inconvenience—one can always retrieve the supply or supplies when the "borrower" makes the mistake of turning his or her own back! What tends to be more of a problem is that inevitably the supplies have been adjusted to different voltage and current settings, so that one then has to go through all the steps of resetting them for the system you are working with.

Quite apart from the problems arising out of "borrowing", there is still the consideration that very often one wants to retain a microprocessor system in the "power up" state for hours at a time, in order to retain a program in the RAM. With most simple systems there is no standby battery supply to maintain the RAM when the main power is removed, and only systems costing thousands of dollars have magnetic discs with automatic "save memory" facilities.

The only alternative with simple systems is to dump out the RAM contents on to paper tape or magnetic cassette, and then load them back in again next time the system is powered up. This can be very tedious and time consuming; hence the attraction of being able to keep the system powered up for reasonable periods of time.

A simple and low cost way to allow this to be done is to make up a fixed-voltage power supply which can be "dedicated" to running the microprocessor system, freeing bench supplies for other uses. This is the approach we have taken, and so far it has worked out well—so well, in fact, that we thought readers might find the supply of interest.

Even if you are a home enthusiast and don't have a problem with "borrowing", you may still find it convenient to build up a supply like ours. If nothing else, it will free your bench supply or supplies for other things—and few hobbyists have all that many bench supplies! If you don't have a bench supply, or don't have enough to run the microprocessor system of your choice, then you will hopefully find the supply even more attractive.

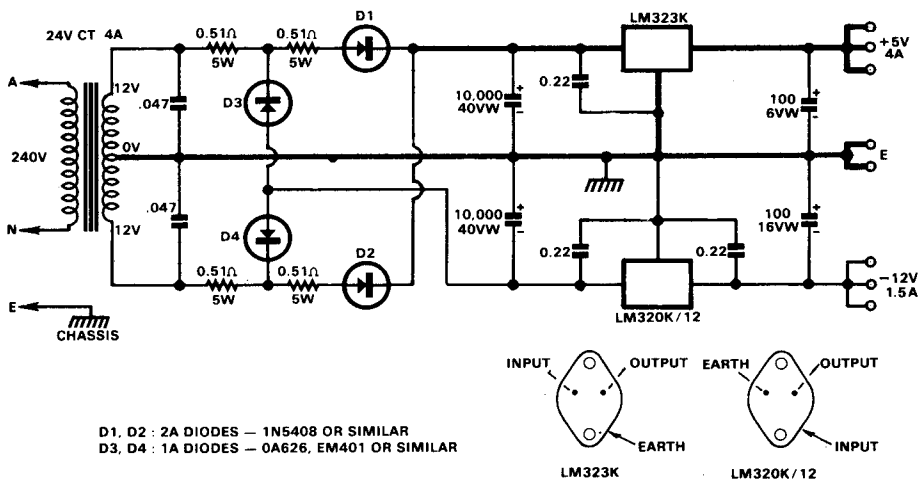
Our supply was designed around parts we had on hand, and because of this it may not be the most elegant design solution. But it works well, and would be an entirely practical proposition if you don't fancy designing one yourself from scratch.

Just about all of the microprocessor evaluation kits and development systems currently available require either a single +5V supply, or a +5V supply and a −12V supply. We have therefore designed the supply as a dual unit, to supply these two voltages.

The +5V supply current requirement is usually somewhat higher than that for the -12V supply, and our supply has been designed with this in mind. The +5V supply will deliver just on 4 amps, while the -12V will deliver about 1.5 amps. In both cases the output is well regulated against both line and load variations, and ripple is typically less than 10mV P-P at loads of 3A and 1A respectively.

As you can see from the circuit, there is nothing particularly special about the supplies. They share a common power transformer, which has a centre-tapped 24V winding rated to supply at least 2A per side. We used a Ferguson type PF 3788, which is actually rated to supply 4A per side, and it accordingly runs almost cold. There is a similar transformer made by A & R, type PT7311. However you could probably get by with a transformer of lower rating, such as the A & R type PT5509 or the Ferguson type PL30/60VA.

Both supplies use full-wave rectifier circuits connected across the transformer secondary. The positive supply uses a pair of 2A diodes, such as the 1N5408 or similar, while the negative supply uses a



D1, D2 : 2A DIODES — 1N5408 OR SIMILAR
D3, D4 : 1A DIODES — 0A626, EM401 OR SIMILAR

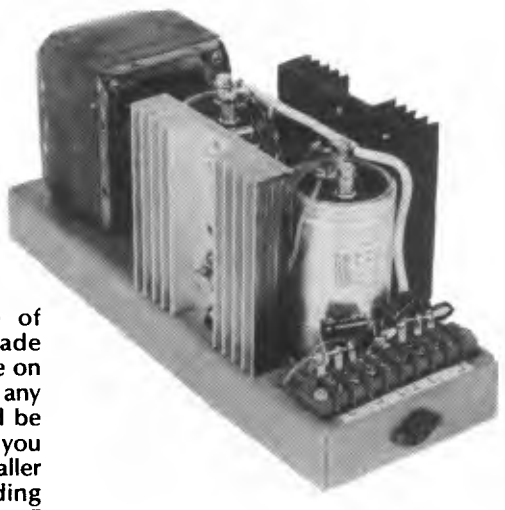
DUAL POWER SUPPLY FOR MICROPROCESSOR SYSTEMS

2/PS/—

pair of 1A diodes such as the 0A626, EM401 or similar.

Each end of the transformer winding is bypassed to ground via a .047uF capacitor to reduce mains transient leakthrough. A 0.51 ohm 5W resistor is also connected in series with each end of the winding, to limit switch-on surge current through the diodes. An additional pair of resistors with the same value are used in series with the positive supply rectifier diodes, to make the rectifier output droop at high currents. This is to reduce the power dissipation of the voltage regulator IC, enabling it to deliver maximum current.

Identical reservoir capacitors are used for both supplies, with a value of 10,000uF. We used two high-grade "computer quality" units which were on hand, and rated at 40VW. However any units rated at 20VW or more would be suitable. For the negative supply you might well be able to use a smaller capacitance value without degrading performance unduly. A value of 4700uF or even 3300uF may be adequate, particularly if you do not anticipate needing the full 1.5A current capacity.



Our prototype supply was built up on long, shallow chassis. Note the heavy wire used for the 5V/4A supply wiring, to minimise voltage drop.

Both supplies are regulated using 3-terminal IC regulators. The positive supply uses an LM323K, while the negative supply uses an LM320K/12. Both of these are in the TO-3 flange-mounting "power transistor" metal package, and are mounted on 100mm square finned heatsinks to limit temperature rise during extended operation.

Each device is fitted with 0.22uF metallised polyester ("greencap") bypass capacitors at its pins to ensure stability. The LM323K requires one between its input pin and common, while with the LM320K/12 it is advisable to fit two—one at the input and one at the output. Each supply is provided with a 100uF electrolytic across the output, to improve transient response.

As you can see from the photograph, we built up the supply on a long narrow chassis salvaged from an earlier project. The power transformer was mounted at one end, with the two large electros mounted in line at the centre flanked by the finned heatsinks for the two regulator ICs. The ICs were actually mounted near the bottom of the heatsinks, to allow for the possible addition of "booster" transistors at some later stage. The transformer terminations, rectifier diodes and protective resistors were mounted beneath the chassis on a length of miniature resistor panel—apart from the mains cord termination, which was made via the recommended screw terminal strip ("B-B" connector).

The outputs from the supply were taken to a screw terminal strip on the top of the chassis, at the end remote from the power transformer. Our strip was eight connectors long, so we allocated three each to the two "active" outputs and two to the earth or "common".

An important point: note that whereas the common connection to the LM323K regulator is brought out to the case, this is not so with the LM320K/12. Here the case is the input terminal, and the common connection is one of the two pins. So that although the case of the LM323K may be mounted directly on the heatsink without insulation, the LM320K/12 must be insulated with the usual washer and plastic bush arrangement. In both cases it is a good idea to use a smear of silicone grease to ensure a good thermal bond.

As mentioned earlier, this supply should be suitable for just about all of the microprocessor systems currently available. To our knowledge there are only two exceptions: the Fairchild and Mostek kits, based on the F8 chip set. Both of these require a +12V supply, as well as the basic +5V supply.

To make the supply suitable for either of these kits, you will need to convert the present −12V supply to one of opposite polarity. This is not difficult to arrange, as follows. Reverse the polarity of D3 and D4, the 10,000uF reservoir electro and the 100uF/16VW output electro. Then in place of the LM320K/12 negative regulator, use an LM340K/12 positive type. This has the same connections as the LM323K, and only requires a single 0.22uF stabilising capacitor between input and common. As the case is common, you won't need to insulate it when mounting. That's basically all there is to it.

SC/MP Tiny BASIC

Back in the December 1976 issue, we announced that National Semiconductor was coming out with a Tiny-BASIC interpreter for their SC/MP, called NIBL. At that stage only a 3k bytes preliminary version was available, with an improved 4k version still to come.

Well, the 4k version of NIBL has now arrived, and it's even better than was predicted. It is now very much an extended Tiny-BASIC, with many powerful features which should make it of great interest and value to professional and hobby computer users alike.

As predicted, it now offers an RND function to generate 16-bit random numbers, and a LINK statement to allow calling machine language subroutines. The hoped-for DO . . . UNTIL statements are also provided, too.

In addition, there is now the ability to perform FOR . . . NEXT loops as in many full BASICs. There is also the ability to handle character strings, and to handle hexadecimal constants. There is also a REM statement for remarks, a MOD function for absolute values, a STAT function to return the current value of SC/MP's status register (allowing the program to manipulate flag and sense lines), and paging functions.

NIBL's formal grammar is now somewhat more flexible, too, allowing greater programming efficiency. Multiple statements per line are now allowed, while LET is no longer mandatory in assignment statements.

And you can now buy NIBL in punched paper-tape form, as an alternative to buying a set of PROMs. A tape costs $15, and may be ordered direct from NS Electronics, Cnr Stud Road and Mountain Highway, Bayswater, Victoria 3153.

Faster SC/MP chip, too

National Semiconductor has also announced a new N-channel version of the SC/MP chip itself. It offers three main features over the existing P-channel chip: Twice the speed, one quarter the power, and only a single +5V supply.

Designated SC/MP-II, the new chip will be available in Australia shortly, from NS Electronics and their various distributors in each state.

First project using a dedicated microprocessor:

ASCII-Baudot translator

As far as we know this project is a world first—the first electronics construction project based on a dedicated microcomputer. Built around a SC/MP evaluation kit, it forms a "black box" which can interface a low-cost surplus Baudot teleprinter to any computer or microcomputer system requiring a 110-baud ASCII teleprinter. It should interest anyone looking for a way of "talking to" one of the new low-cost microcomputer systems at low cost.

by JAMIESON ROWE

Like most small mini-computers, most of the microcomputer evalution kits and development systems currently becoming available have been designed to converse with the user via a computer-type teleprinter such as the well-known Teletype model ASR-33. Communication is via 20mA current loops, at a rate of 110 bauds (bits per second), and in the ASCII code.

While teleprinters of this type are available fairly readily in the USA, where most of the microprocessor systems originate, they are not at all easy to come by in Australia. New, they are likely to cost you something like $1500 plus tax—hardly within the grasp of the average small user or would-be computer hobbyist, and rather out of proportion to the current cost of a typical microprocessor system.

Very few machines have appeared from time to time on the second-hand market, but even these have been relatively expensive. You could expect to pay anything from $400 up, depending upon condition. This is still rather a lot to pay if one wants to use it to talk to a microcomputer kit costing around $100-200!

In contrast with these computer-type teleprinters, quite good supplies of the older Baudot-type teleprinters are currently available at much lower cost, from firms dealing in surplus equipment. Machines of British, German and US manufacture are available, most of them having been sold as obsolete plant by the Army, Telecom and other public utilities.

For example you can currently pick up a Teletype model 15 page teleprinter for around $100, or a model 14 typing reperforator for about half that price. Similarly, Creed model 7A page teleprinters are available at about $120.

These machines are obviously quite attractive in terms of price, compared with the computer-type machines. However, they are not directly compatible with the majority of computer systems, for a number of reasons. The most obvious of these is that they use the 5-level Baudot code, instead of the 7-level ASCII code.

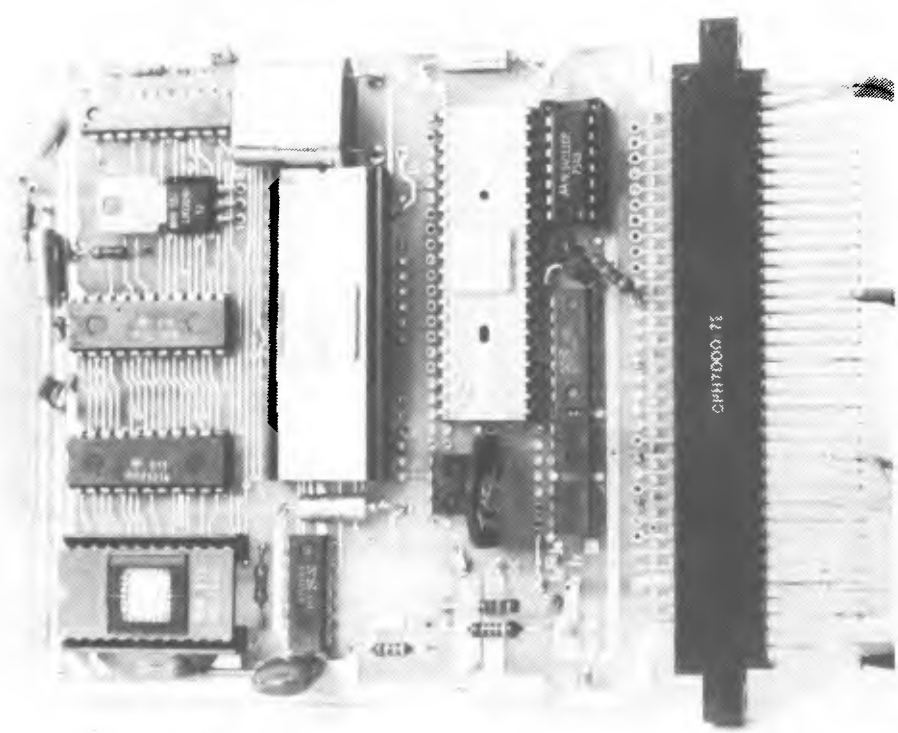
In itself, this is not an insurmountable problem. Code translation can be performed using suitable ROMs, although even this is not as simple as one might expect. This is because Baudot code is really a 6-bit code in which the normally missing 6th bit is effectively sent only when it changes value. It is sent in encoded form as special "letters" and "figures" characters, so that any code conversion system must be capable of performing the appropriate storage and "housekeeping" functions.

Quite apart from the problems associated with code translation as such, there is also another problem: speed. Computer-type machines mostly run at 110 bauds, and with a total of 11 bits per character, this gives a maximum character rate of 10 per second. In contrast, Baudot machines typically run at 50 bauds, and with a total of 7.5 bits per character, this gives a maximum character rate of around 6.5 per second.

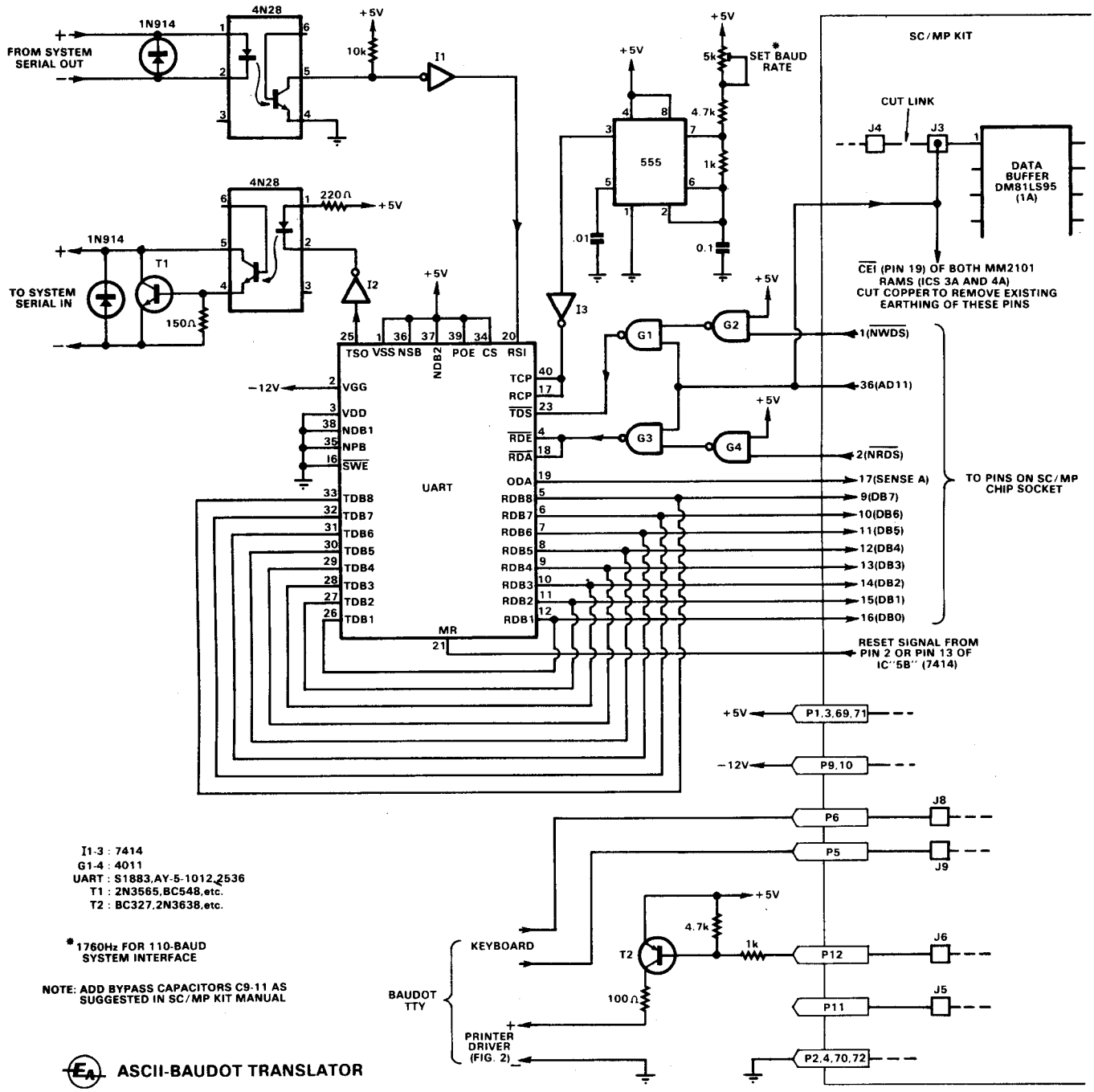
In fact, because of the need to send the special "LTRS" and "FIGS" characters interspersed with the actual characters, the maximum character rate of 50-baud Baudot machines in practice is nearer 5 per second—less than half that of a 110-baud computer teleprinter.

The difference in speed doesn't pose a problem for "inward" data flow, from the teleprinter keyboard to the computer system, because here the speed differential assists rather than opposes. However, the problem comes with "outward" data flow, from the computer system to the teleprinter print mechanism. The system is geared to output characters at a rate of 10 per second, while the Baudot printer can only "digest" them at a rate of about 5 per second.

As a result, it becomes necessary to provide temporary storage or "buffering", between the system output and the printer. The buffer must be of the "first-in-first-out" (FIFO) type, capable of accepting characters at the high rate,



A view of the ASCII-Baudot translator, built up on the SC/MP kit PC board.



ASCII-BAUDOT TRANSLATOR

supplying them at the slow rate, and accumulating the difference as required.

Hopefully you can see from all this that interfacing a surplus Baudot teleprinter to a microcomputer system isn't exactly the proverbial "piece of cake". There are quite a few functions to be performed, and to do the job with a conventional wired logic circuit would involve quite a lot of ICs and a complex PC board.

Happily, thanks to modern IC technology there is now an easier and more elegant way of doing the job. This is to use one of the low-cost microprocessor evaluation kits currently available, and turn it into a dedicated "black box".

This is the approach I have taken, and the ASCII-Baudot translator which will now be described is based on the SC/MP evaluation kit currently selling for around $80 plus tax.

On the hardware side, most of the functions of the translator are performed by the basic SC/MP kit circuitry. However, six additional ICs are required, along with two transistors and a hand-full of minor components. Most of these fit on the spare space provided on the SC/MP kit PC board.

The main supplementary IC required is a UART (universal asynchronous receiver-transmitter), which performs the 110-baud ASCII interfacing to the main computer system. The UART may be an S1883 (American Micro-systems Inc), an AY-5-1012 (General Instruments), a 2536 (Signetics), or any other exactly equivalent device. Note that not all UARTs currently available are exactly equivalent to these, however.

A 555 timer IC is used to generate the 1760-Hz clock signals required by the UART. Two 4N28 or similar low-cost opto-couplers are used to perform the actual interfacing to the main computer, to ensure that the translator is fully compatible with all systems designed for 20mA current loop signals. Transistor T1 is used to boost the current switching ability of the outgoing opto-coupler.

Schmitt trigger elements I1, I2 and I3 are used for inversion and signal squaring, while gates G1-4 are used to perform address decoding for the UART so that it effectively occupies location FFF in the SC/MP memory space. Transistor T2 is used to perform interfacing between the

existing serial output of the SC/MP kit and the driver in the Baudot teleprinter.

Most of the signals required for the additional circuitry are taken from various pins on the SC/MP chip socket, on the kit PC board. However, there are also three small modifications which must be made to the basic kit board. All three are to ensure that the RAMs and ROM are disabled whenever SC/MP addresses the UART.

One of the three modifications is quite straightforward. It involves cutting the link provided in the PCB pattern between pads J3 and J4, adjacent to pin 1 of the DM81LS95 data buffer IC. A wire is then added to the board so that it connects pin 1 of the IC to pin 36 of the SC/MP socket. This causes the data buffer to be disabled whenever address bit AD11 is high, corresponding to the UART being addressed.

The other two modifications are similar, but a little more tricky in the mechanical sense. The PCB copper pattern on the kit normally earths pin 19 of both MM2101 RAMs, but the translator uses these pins to prevent the RAMs from writing when the UART transmitter is addressed. Accordingly the copper laminate should be removed from around the pin 19 hole for both ICs (preferably before the ICs are mounted on the PCB). This allows the pins to be wired as before to pin 36 of the SC/MP socket, so that both RAMs are disabled when the UART is addressed.

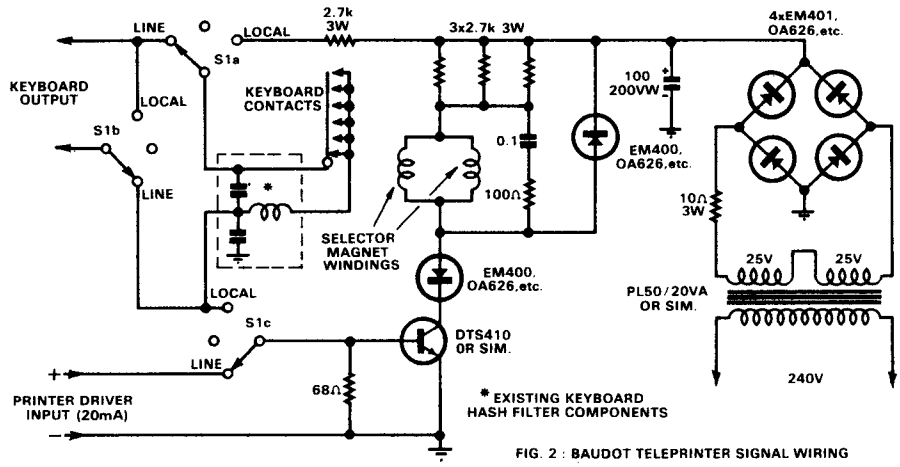


FIG. 2 : BAUDOT TELEPRINTER SIGNAL WIRING

The only other change required to the basic SC/MP kit to convert it into the translator is replacement of the original 512-byte ROM containing "Kitbug", with a pin-compatible EPROM containing the ASCII-Baudot translator program I have written. The EPROM required is the National Semiconductor MM5204, and it may be ordered from the various NS distributors with the translator program resident.

The program is actually "Mark 7", or the seventh version—it took a while to come up with a program which did all the right things at the right times, and would interface to all systems!

Actually I obtained good results with "Mark 4", but it used the reverse interfacing scheme to that shown here—the UART was used for interfacing on the Baudot side, with the SC/MP program performing the ASCII side interfacing. The only problem with this arrangement was that it wasn't capable of receiving an ASCII character while it was sending one; this caused problems when it was connected to computer systems which "echo" keyboard input on a bit-by-bit basis rather than on the more usual whole-character basis. So I finally elected to reverse the system of interfacing, and rewrite the whole program, in order that the translator would be fully compatible with all systems.

The complete final program fits into 512 bytes, including its lookup table of ASCII-Baudot code equivalents. It uses the first six addresses in the SC/MP kit RAMs as a "stack", for storage of counters and pointers, leaving the remaining 250 locations in the RAMs as the character buffer. Since the buffer only accumulates characters according to the difference between the ASCII input and Baudot output rates—i.e., at about 5 characters per second, this gives the translator the capacity to cope with quite respectable character strings.

A complete listing of the program in symbolic form would be far too long to permit publication here. However, I am publishing a full hexadecimal listing, for the benefit of those who may wish to program their own PROMs.

The program itself starts at location 001, which is where SC/MP normally fetches its first instruction upon being powered up. Hence the translator needs no deliberate initialisation by the user—all one does is turn on the power, and away it goes.

Most of the Baudot teleprinters available on the surplus market have printer selector magnets with dual windings, which were intended for either series operation in nominal 20mA loops, or parallel operation in nominal 60mA loops.

The windings have quite high inductance, however, and are not suitable for

Here is complete hex listing of the translator program, resident in an EPROM.

```
0000 08 04 C4 02 35 C4 00 31 C4 00 37 C4 45 33 C4 06
0010 C9 04 C4 07 C9 05 C4 07 C9 06 06 D4 FE 07 C4 40
0020 C9 02 90 24 06 D4 10 98 18 C4 02 36 C1 05 32 C0
0030 CF D4 7F CE 01 A9 05 9C 04 C4 06 C9 05 C4 36 90
0040 02 C4 78 8F 09 3F 90 DC 06 D4 10 98 1A C4 02 36
0050 C1 05 32 C0 AB D4 7F CE 01 A9 05 9C 0A C4 06 C9
0060 05 90 04 C8 9B 90 6C 06 D4 20 9C 67 C4 10 C9 00
0070 C4 00 01 3F 06 D4 20 9C 5A 3F 3F 06 D4 20 98 02
0080 C1 00 01 58 01 C1 00 1C C9 00 9C ED 3F 40 E4 02
0090 9C 04 C4 0D 90 CD 40 E4 08 9C 04 C4 0A 90 C4 40
00A0 E4 04 9C 04 C4 20 90 BB 40 98 B8 E4 1F 9C 04 C9
00B0 01 90 20 40 E4 1B 9C 06 C4 20 C9 01 90 15 C4 01
00C0 36 C4 6E 32 C1 01 58 01 C6 02 98 97 60 9C F9 C2
00D0 FF 90 90 C1 05 03 F9 04 98 7A C4 02 36 C1 04 32
00E0 C2 00 01 40 E4 0D 9C 04 C4 02 90 42 40 E4 0A 9C
00F0 04 C4 08 90 39 40 E4 20 9C 04 C4 04 90 30 40 E4
0100 40 9C 04 C4 2B 90 27 40 98 24 D4 40 E1 02 98 0F
0110 40 D4 40 C9 02 98 04 C4 1F 90 1E C4 1B 90 1A C4
0120 01 36 C4 6F 32 C6 02 98 05 60 9C F9 C2 FD 01 A9
0130 04 9C 07 C4 06 C9 04 90 01 01 06 DC 01 07 C4 10
0140 C9 00 3F 3F 40 D1 00 98 05 06 D4 FE 90 03 06 DC
0150 01 07 90 02 90 10 C1 00 1C C9 00 9C E5 3F 3F 06
0160 D4 FE 07 3F 3F 3F C4 00 36 C4 47 32 3E 08 2D 30
0170 3D 31 39 32 30 33 2A 34 21 35 35 36 3C 37 2C 38
0180 23 39 18 41 13 42 0E 43 12 44 10 45 16 46 0B 47
0190 05 48 0C 49 1A 4A 1E 4B 09 4C 07 4D 06 4E 03 4F
01A0 0D 50 1D 51 0A 52 14 53 01 54 1C 55 0F 56 19 57
01B0 17 58 15 59 11 5A 33 3F 32 24 3E 28 29 29 3A 07
01C0 37 2F 38 2D 2E 3A 2F 3D 31 2B 34 27 27 2E 26 2C
01D0 2B 40 36 25 25 26 32 2A 37 3B 18 61 13 62 0E 63
01E0 12 64 10 65 16 66 0B 67 05 68 0C 69 1A 6A 1E 6B
01F0 09 6C 07 6D 06 6E 03 6F 0D 70 1D 71 0A 72 00 00
```

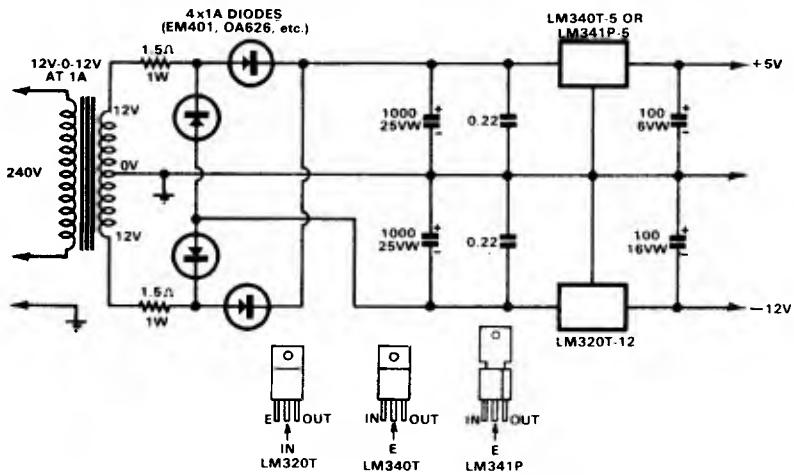


FIG. 3: POWER SUPPLY FOR TRANSLATOR

direct connection into low-voltage circuitry. They should be operated from a substantially constant-current source in order to obtain reliable character decoding. This generally means operating with the windings in parallel, and from a current source employing at least 50 volts. Accordingly a driver transistor must be used, to allow the low-voltage circuitry in the translator to control the magnet current.

Fig. 2 shows the power supply and driver circuit I have built into my machine, a Teletype model 15. It works very well, yet uses only a few easily obtained parts. The driver transistor is a Delco DTS410, but any similar high-voltage medium-power NPN transistor would be suitable. It is operated in switching mode, and does not dissipate significant power.

The circuit of Fig. 2 shows switching which allows the keyboard and printer driver to be connected together for "local" operation, if desired. This is not essential, but I have found it convenient.

Note that the input drive current required by the driver stage is a nominal 20mA, so that it would also be quite suitable for use with solid-state radio-teletype demodulators.

Fig. 3 shows the circuit of a small power supply suitable for the translator itself. It is quite straightforward, and uses two low-power 3-terminal regulator devices to provide the +5V and -12V supplies required. It would of course be possible to power the translator from the power supply used for the main computer system, if it has the capacity to supply around 400mA at +5V and 250mA at -12V additional to its existing loading.

If your Baudot teleprinter is one of the Teletype machines, you will need a step-down transformer to run its 115V motor from the 240V mains. The motor draws about 1.2 amps, calling for a transformer with a rating around 140VA. A double-wound type is recommended because the insulation of the motor may not be capable of withstanding 240V. I have used a Ferguson Transformer type TS115/125, which although nominally rated at 125VA, runs quite cool even after many hours of continuous operation.

An important point to note if you are using the translator with Teletype machines is that the printing mechanism of these machines is fitted with an optional "down-shift on space" facility. This MUST be disabled for use with the translator, or the translator and printer will get out of step in terms of letters-figures shifting.

On the page teleprinters, models 15 and 19, the facility is controlled by a small horizontal lever at the very bottom of the printer mechanism, right at the front and centrally placed beneath the horizontal selector vanes which convey the character code to the moving type carriage.

To disable the automatic downshift, the keyboard should be removed by undoing the two knurled thumbscrews on each side, and then sliding it out the front. Then using a short screwdriver, loosen the locking screw on the down-shift lever and move its rearward end towards the left until it hits the stop. Finally, tighten the lockscrew again, and replace the keyboard.

On the model 14 typing reperforator the automatic downshift facility is controlled by the position of the vertical selector bar at the extreme left-hand end of the fixed type basket, looking from the front. To disable the downshift, the bar must be moved from the deeper of the two available slots, on the right, to the shallow slot on the left. This prevents it moving when a "space" character is decoded. The bar can be moved by hand; no tools are necessary.

Before closing, there are a few comments which should be made.

If you get the translator going and it seems to make consistent printing errors with some of the figures-case characters, this will almost certainly be due to your Baudot teleprinter having coding which differs from that assumed by the translator.

```
LOWER CASE (LTRS):  ABCDEFGHIJKLMNOPQRSTUVWXYZ
UPPER CASE (FIGS):  -?:$3%@£9 ().,$014'57=2/6+
```

Fig. 4: Both cases of the Baudot character set assumed by the translator program.

This possibility exists because Baudot machines are not all consistent in terms of upper-case characters and coding. Most of the machines available in Australia use the so-called "International Telegraph Code No. 2", and this is the code I have assumed in designing the translator. However you may be unlucky enough to get a machine with different coding, in which case you will get printing errors. They will be consistent, however, so that once you get used to them they should be only a minor irritation.

You can check the coding of your

This low cost hexadecimal keyboard unit for use with the National SC/MP available from National Semiconductor.

machine by either looking at the type-bar heads, or running it in local and typing out each of the keys in alphabetic order, in both lower case (LTRS) and upper case (FIGS.) Either way, you can compare the results with the coding type-out shown in Fig. 4, which is that assumed by the translator.

The translator itself does make one minor decoding error, which proved difficult to obviate. It affects only the "at" or "per" sign (@), occurs only during ASCII-to-Baudot translation (i.e., during printing), and even then not every time.

You will find that whenever the character occurs immediately following figures or other upper-case Baudot characters, it will be translated and printed correctly. However, when it occurs immediately following letters, you will find that it will be mis-translated and printed as a G—its lower-case Baudot equivalent.

The reason for this error is that in ASCII code, the "at" sign is effectively an alphabetic character—it has the same coding for the three most significant bits as do the first 15 letters. However, in Baudot code, where the character is present, it is regarded as an upper-case character or "figure".

Full processing of the character by the translator program would involve not only recognising and translating its code, but also performing the Baudot case checking and modification procedure which the program does with all other printing characters. This would involve sending the appropriate "FIGS" character ahead of the character itself, if the printer happened to be in letters mode.

This proved to be rather difficult, so I ended up compromising. The "at" sign is correctly translated into the appropriate 5-bit Baudot code, but is otherwise treated like one of the non-printing characters like space, carriage return or line feed. This means that the program does not perform the usual case checking or modification, because in Baudot code the non-printing characters are "caseless"—they are the same in both cases, and hence do not affect printer case mode.

When you're using the translator and a Baudot machine with a system, remember that the translator buffer can only cope with an accumulation of 240 characters. So if you want the system to print out a long hex listing or some text, do it in modest slabs.

If you forget and try printing out a large character string, you'll find that the translator will "lose" 240-character chunks of the string, due to the buffer input pointer having overtaken the output pointer. This causes no problem to the translator, but it does foul up your print-out! So if you find that chunks of a printout are missing, simply try again with the system delivering the characters in smaller strings.

By the way, the use of a code translator inevitably involves a time delay in either direction—mainly because characters must be fully strobed in before they can be translated and then sent out again. This means that when characters are keyed in from the teleprinter keyboard there is a noticeable delay before they are echoed by the printer. The delay is particularly noticeable when the computer system itself echoes on a complete-character basis, giving a total "round trip" delay time of around 450 milliseconds.

This takes a little getting used to, but it is one of the prices one pays for using a teleprinter which "speaks a different language".

Finally, a word on using Baudot teleprinters with computers. Baudot machines were developed around 1906, long before electronic digital computers were a reality. Presumably their designers didn't anticipate that anyone would ever want to use them to send lots of mixed letters and numerals, so they adopted the system of using "letters" and "figures" modes, with the two special case-change keys used to change back and forth between modes as required.

It won't take you long to realise that this makes the machines quite clumsy when it comes to keying in programs in hexadecimal code. You'll have to develop the habit of always sending "LTRS" before you want to key in letters, and always sending "FIGS" before keying in figures.

The technique takes a while to get used to, and even then it is mildly irritating—especially if you are used to working with the more elegant ASCII machines. Still, Baudot machines have one big advantage—they're available, and they're cheap!

ASCII-Baudot translator — notes

ASCII-BAUDOT TRANSLATOR (October 1976, File No. 2/CC/15): Philips Industries Ltd has advised that the Signetics 2536 UART is no longer available. However they are able to supply the TMS 6011, which is an exact equivalent device.

ASCII-BAUDOT TRANSLATOR (October 1976, File No. 2/CC/15): It was not stated explicitly that the translator was designed to operate at 50 bauds. This is in fact so, and the teleprinter must accordingly be set at this rate for correct operation.

```
10 PRINT "HI, I WORK OUT FACTORIALS."
20 PRINT "WHAT NUMBER WOULD YOU LIKE";
30 INPUT N
40 LET F=1
50 IF N<=1 THEN GOTO 90
60 LET F=F*N
70 LET N=N-1
80 GO TO 50
90 PRINT "ITS FACTORIAL IS",F
91 PRINT "DO YOU WANT TO WORK OUT ANOTHER (1=YES,0=NO)";
92 INPUT A
93 IF A=1 THEN GO TO 20
100 END

>RUN
HI, I WORK OUT FACTORIALS.
WHAT NUMBER WOULD YOU LIKE? 6
ITS FACTORIAL IS 720
DO YOU WANT TO WORK OUT ANOTHER (1=YES,0=NO)? 1
WHAT NUMBER WOULD YOU LIKE? 7
ITS FACTORIAL IS 5040
DO YOU WANT TO WORK OUT ANOTHER (1=YES,0=NO)? 0
```

Here are the two simple Tiny-BASIC programs which the author wrote to try out the pre-release version of NIBL. One works out factorials, the other is a number game.

useful subset. Here's a summary of what you get:

Valid statement forms include INPUT (for numbers only), LET, GO TO, GO SUB and RETURN, IF THEN, and PRINT. There is also a CALL statement, to allow calling machine-language subroutines. The latter facility is very valuable, of course, because it will allow NIBL to be expanded.

The final version of NIBL may also have the ability to interpret DO . . . UNTIL statements, if PROM space permits.

Program control statements provided are LIST, RUN, END, and CLEAR. The first of these may be used to list either the whole program, or alternatively a single line. An inbuilt editor allows lines to be replaced, and also additional lines added as required. All that is necessary for correct execution is that lines are numbered consecutively between 1 and 32, 767.

If statement lines begin with a number, NIBL stores them away as a program for deferred execution. If a statement is not numbered, NIBL executes it immediately upon entry (following the user terminating the line with a carriage return).

NIBL is only capable of performing integer arithmetic, on numbers within the range from −32,768 to +32,767. It provides the four basic arithmetic functions, represented by the symbols +,−,* and /, together with the logic operators AND, OR and NOT, and a 16-bit random number generator function called by the label RND. Constants may be expressed in either decimal or hexadecimal.

Up to 26 variables may be used in a NIBL program, using single alphabetic characters as labels (A-Z inclusive). Parenthesis is permitted, and subroutines may be nested to 16 levels.

All of the normal relational operators are provided, including equals, greater than, less than, greater than or equals, less than or equals, and not equal to.

NIBL also provides an operator of indirection, symbolised by the "at" or "@" sign. When placed immediately preceding a variable this causes the variable to be interpreted as a decimal address in SC/MP memory space. If A is a variable with value 256, the statement LET B=@A gives variable B the value equal to the data byte in decimal memory location 256.

What is NIBL like in practice? Well, at the time of writing the final version was not yet available in Australia, but thanks to NS Electronics I was able to try an earlier pre-release version. This was in only 3k of PROMs, and didn't have some of the features which will be in the full 4k version—like the RND function or the logic functions, or the CALL statement.

However it was certainly very interesting to try the smaller version out, with one of the SC/MP LCDS systems. Even though its facilities were rather limited, it was still very nice to be programming at a higher level of abstraction and be able to make corrections on-line.

In fact after having got a couple of simple programs up and running, it was with surprise that I noticed the time: less than an hour after the NIBL card had been plugged in and the system turned on!

As the two programs might be of interest to readers, I have reproduced them on these pages. In each case the program itself is listed first, followed by a sample of the execution.

As you can see, the first program is a very simple one which calculates the factorial of a number fed in from the keyboard. It prompts the user for a number, prints out the answer and then asks if the user wishes to work out another. A negative reply causes it to halt.

```
10 PRINT "HI! I WILL THINK OF A NUMBER BETWEEN 0 AND 255."
20 PRINT "WHEN I HAVE,TRY TO GUESS ITS VALUE. (I WILL HELP)"
30 LET B=1
40 LET A=@B
50 PRINT "OK, I HAVE A NUMBER"
60 PRINT "WHAT IS YOUR GUESS";
70 INPUT C
80 IF C=A THEN GOTO 140
90 IF C>A THEN GOTO 120
100 PRINT "TOO SMALL. NEXT GUESS";
110 GO TO 70
120 PRINT "TOO BIG. NEXT GUESS";
130 GO TO 70
140 PRINT "YOU GUESSED IT!!!   LET'S PLAY AGAIN."
150 LET B=B+1
160 GO TO 40
170 END

>RUN
HI! I WILL THINK OF A NUMBER BETWEEN 0 AND 255.
WHEN I HAVE,TRY TO GUESS ITS VALUE. (I WILL HELP)
OK, I HAVE A NUMBER
WHAT IS YOUR GUESS? 200
TOO BIG. NEXT GUESS? 180
TOO SMALL. NEXT GUESS? 196
YOU GUESSED IT!!!   LET'S PLAY AGAIN.
OK, I HAVE A NUMBER
WHAT IS YOUR GUESS? 128
TOO BIG. NEXT GUESS? 64
TOO BIG. NEXT GUESS? 32
TOO BIG. NEXT GUESS? 0
TOO SMALL. NEXT GUESS? 16
YOU GUESSED IT!!!   LET'S PLAY AGAIN.
OK, I HAVE A NUMBER
WHAT IS YOUR GUESS? 128
TOO BIG. NEXT GUESS? 0
TOO SMALL. NEXT GUESS? 64
TOO BIG. NEXT GUESS? 32
TOO SMALL. NEXT GUESS? 45
TOO SMALL. NEXT GUESS? 56
TOO BIG. NEXT GUESS? 60
TOO BIG. NEXT GUESS? 50
TOO SMALL. NEXT GUESS? 54
YOU GUESSED IT!!!   LET'S PLAY AGAIN.
OK, I HAVE A NUMBER
WHAT IS YOUR GUESS?
! 7 AT 70
>
```

Incidentally, this little program soon comes up against the inability of NIBL to cope with numbers greater than 32,767. In fact the largest number it can find the factorial for is 7; 8 causes overflow.

The longer program is a simple number guessing game. As the pre-release version of NIBL didn't have the RND function, I had to use the indirect operator to generate pseudo-random numbers by fetching instruction bytes from NIBL's own PROMs! As you can see, this worked fairly well.

The program can be quite good fun, giving you a taste of the appeal in computer games.

Of course with the final version of NIBL, it will be possible to run games like this which will be rather more satisfying, using the RND function to generate less predictable numbers. In fact quite a few games have been written in Tiny-BASIC, and should be capable of being run with NIBL.

In short, NIBL seems to be very good news for SC/MP users.

You'll be able to order NIBL from NS distributors throughout Australia. For further information, contact NS Electronics at either Cnr. Stud Road and Mountain Hwy, Bayswater, Victoria 3153, or 2-4 William Street, Brookvale, NSW 2100.

Software/Firmware development—2

A Text Editor for SC/MP

Here is a symbolic text editor program which the author has written for National Semiconductor's SC/MP low cost development system. It provides all the basic text editing functions to let you prepare programs for assemblers, compilers, etc. You will be able to buy it resident in 1k bytes of PROM and ready to go, or, alternatively, it can be fed into your system via tape or cassette, and run in RAM.

by JAMIESON ROWE

If you've ever tried punching up a paper tape of a program in assembly or problem-orientated language using a teleprinter on "local", you'll know just how frustrating it can be. Even if you are extremely careful it seems to be impossible not to make a few errors, and Murphy's Law always seems to ensure that you rarely discover these until you have typed in at least three more lines!

Of course if you realise that you made an error immediately after having done it, you can use the back-space facility and delete the wrong character(s) with the rubout key. But if you don't discover an error until later on, you are forced to either perform cut-and-paste surgery on the tape, or try stop-and-go editing of the tape while punching a new "clean" version.

Preparing long problems in this way can be very tedious and time consuming. Small wonder, then, that the people working on minicomputers and larger machines have for years been using the computer itself to make the job easier and faster. This is done by using a software utility program known as a symbolic text editor.

Using such a program, you type your own program text into a section of the computer's memory which is set aside as the "buffer". Then with the text in the buffer, the editor lets you change lines, delete lines, insert extra lines at any desired point, inspect lines or groups of lines, and finally either type out a listing or punch out a clean tape (or both).

In short, a text editor program can be a very useful item of software, and it is well worth having one even on small microcomputer systems. The only trouble is that not too many microcomputers have been provided with editor programs as yet, particularly the systems based on the more recent microprocessor chips.

To help alleviate this situation, I have written a text editor program for the National Semiconductor SC/MP lowcost development system, as described in our October issue. I chose this system because at present the SC/MP chip and its systems appear to be growing fastest in popularity, particularly in the hobby area.

The editor program itself is written in SC/MP machine language, and occupies 864 bytes of memory. It uses the 256-byte RAM in the LCDS system base as a stack and line address buffer, but can use RAM memory at any other location in SC/MP memory space for its text buffer. The larger the available RAM, the more text the editor can handle at one time.

I have arranged for the editor to be available from NS Electronics distributors, written into a pair of MM5204 512-byte PROMs. With the PROMs plugged into the correct locations on a SC/MP ROM card which is programmed for the appropriate address range (hex. 3000-3FFF), you will then have the editor permanently resident in your system, and available at any time merely by calling it at its starting address (hex. 3C00).

Alternatively, you can run the editor in RAM memory, loading it in each time you need it by means of a punched paper tape or cassette. I am reproducing a full hexadecimal listing of the program on these pages, to allow you to do this if you wish. The four-digit numbers at the start of each line are addresses; as you can see the program runs from 3C00 to 3F5F, inclusive.

Note that if you do elect to run the editor in RAM, you will need at least one 2k-byte RAM card. This will give you 1k of RAM left for the text buffer—enough for small text slabs, but barely good enough for serious work. With the editor in PROMs, even a single RAM card gives you a full 2k for the text buffer, which is very much more practical.

When the editor is called, it announces itself and then asks you to give the available text buffer range in memory. This must be supplied as two hexadecimal numbers, separated by a non-hex character such as a space or hyphen. Leading zeroes are not required, but the second number must end with a carriage return.

The editor then enters its command mode, signalling this by ringing the bell. The user may then type in a command letter, followed by a carriage return. If text is to be fed in via the keyboard, the command letter "A" is appropriate, while "R" tells the editor to read in a previously punched tape via the reader.

```
SC/MP SYMBOLIC EDITOR PROGRAM.
WRITTEN BY J.ROWE, ELECTRONICS AUSTRALIA
FOR SC/MP L.C.D.S. SYSTEMS

BASIC COMMANDS AND THEIR FUNCTIONS:
(THE CLOSING BRACKET ")" SYMBOLISES A CARRIAGE RETURN)

COMMAND                 FUNCTION

A)                      APPEND LINES TO BUFFER
R)                      READ TAPE INTO BUFFER
L) . ML) , M,NL)        LIST ALL LINES, OR LINE M, OR LINES M-N
MC) , M,NC)             CHANGE LINE M, OR LINES M-N
MI)                     INSERT LINE OR LINES BEFORE LINE M
MD) , M,ND)             DELETE LINE M, OR LINES M-N
K)                      KILL TEXT IN BUFFER
P) , MP) , M,NP)        PUNCH ALL LINES, LINE M, OR LINES M-N
B) , MB) , M,NB)        AS FOR PUNCH, BUT PUNCHES A BELL CHAR
                        AT END OF TEXT
/                       EDITOR PRINTS NUMBER OF LINES CURRENTLY
                        HELD IN TEXT BUFFER, IN DECIMAL
BELL                    WHEN IN TEXT MODE, RETURNS EDITOR TO
                        COMMAND MODE
(M AND N REPRESENT DECIMAL LINE NUMBERS)
```

Here is the basic command set of the editor, showing the various command letters, the possible arguments for each, and their functions. In text mode, a percent sign acts as a backspace.

```
3C00 04 C4 77 36 C4 FF 32 C4 7B 37 C4 16 33 3F 3F 07
3C10 C4 4F 33 3F C6 01 CA F1 C6 01 CA EF 3F C6 01 CA
3C20 EF C6 01 CA ED C4 00 CA F9 C2 F0 CA F8 C2 EF CA
3C30 F7 C4 7A 37 C4 E1 33 C4 07 3F C4 0D 3F C4 0A 3F
3C40 C4 00 CA F4 CA F3 CA F2 C4 7A 37 C4 90 CA F1 33
3C50 3F D4 7F 01 C4 3F 35 C4 2B 31 C5 03 98 3B 60 9C
3C60 F9 C1 FE CA F6 C1 FF 31 C2 F6 35 C2 F9 03 FA F3
3C70 94 02 90 12 C2 F9 03 FA F2 94 02 90 09 C2 F2 98
3C80 10 03 FA F3 94 0B C4 7A 37 C4 E1 33 C4 3F 3F 90
3C90 A9 3F D4 7F E4 0D 9C EE 3D 40 D4 70 E4 30 98 19
3CA0 40 E4 2F 9C 09 C4 3E 35 C4 B2 31 3D 90 8C 40 E4
3CB0 2C 9C D3 C4 01 CA F4 90 8F 40 D4 0F CA F5 C2 F4
3CC0 9C 04 C2 F3 90 02 C2 F2 02 01 40 70 01 70 70 01
3CD0 40 70 F2 F5 01 C2 F4 9C 05 40 CA F3 90 D9 40 CA
3CE0 F2 90 D4 35 CA F6 31 CA F5 C2 F7 CA E9 C2 F8 CA
3CF0 EA C2 F1 E4 84 98 09 C4 7A 37 C4 E1 33 C4 0A 3F
3D00 C4 7A 37 C2 F1 33 C2 EE 02 F4 01 E2 F8 98 53 C2
3D10 F8 35 C2 F7 31 3F D4 7F 01 40 E4 0A 98 1D 40 E4
3D20 07 98 3F 40 E4 25 9C 04 C5 FF 90 0F 40 98 0C E4
3D30 7F 98 08 40 E4 0D 98 01 40 CD 01 C4 77 35 CA F8
3D40 C2 F3 02 F2 F3 31 CA F7 40 E4 0D 9C B6 C2 E9 C9
3D50 00 C2 EA C9 01 C2 F9 E4 74 98 07 C2 F6 35 C2 F5
3D60 31 3D C4 3C 35 C4 30 31 C4 90 CA F1 3D 08 08 08
3D70 C4 80 90 02 C4 01 CA F1 C2 F3 9C 0A C4 01 CA F3
3D80 C2 F9 CA F2 90 08 C2 F2 9C 04 C2 F3 CA F2 C4 E1
3D90 33 C4 0A 3F C2 F1 98 15 C4 7A 37 C4 88 33 3F C4
3DA0 E1 33 C4 C0 CA F6 C4 00 3F AA F6 9C F9 C4 77 35
3DB0 C2 F3 02 F2 F3 31 C1 00 CA F6 C1 01 35 C2 F6 31
3DC0 C5 01 98 03 3F 90 F9 C4 0D 3F C4 0A 3F C2 F3 E2
3DD0 F2 98 04 AA F3 90 D6 C2 F1 98 1A 94 03 C4 07 3F
3DE0 C4 C0 CA F6 C4 00 3F AA F6 9C F9 C4 7A 37 C4 88
3DF0 33 3F C4 E1 33 C4 3C 35 C4 39 31 3D 08 08 08 08
3E00 C2 F2 9C 06 C2 F3 98 35 CA F2 BA F3 C2 F2 E2 F9
3E10 98 20 AA F3 AA F2 C4 77 35 C2 F2 02 F2 F2 31 C4
3E20 77 37 C2 F3 02 F2 F3 33 C1 00 CB 00 C1 01 CB 01
3E30 90 DA C2 F3 CA F9 C4 3C 35 C4 30 31 3D C4 3C 35
3E40 C4 83 31 3D C2 F3 98 F5 C2 F2 9C F1 C4 77 35 C2
3E50 F3 02 F2 F3 31 C1 00 CA EB C1 01 CA EC C4 3C 35
3E60 C4 E2 31 3D C2 F9 CA F2 AA F9 C4 77 35 C2 F2 02
3E70 F2 F2 31 C1 00 C9 02 C1 01 C9 03 C2 F2 E2 F3 98
3E80 04 BA F2 90 E5 C2 EB C9 02 C2 EC C9 03 AA F3 90
3E90 BB C2 F2 9C 06 C2 F3 98 A4 CA F2 C4 3C 35 C4 E2
3EA0 31 3D C2 F3 E2 F2 98 04 AA F3 90 EF C4 3C 35 C4
3EB0 30 31 3D C4 00 CA F6 CA F5 CA F4 C2 F9 02 F4 9C
3EC0 94 05 C2 F9 01 90 03 01 AA F6 40 02 F4 F6 94 02
3ED0 90 05 01 AA F5 90 F3 40 02 F4 FF 94 02 90 05 01
3EE0 AA F4 90 F3 C4 7A 37 C4 E1 33 C4 3D 3F C2 F6 98
3EF0 03 C4 31 3F C2 F5 98 03 DC 30 3F C2 F4 DC 30 3F
3F00 C4 3C 35 C4 39 31 3D 0D 0A 45 44 49 54 4F 52 20
3F10 52 45 41 44 59 2E 0D 0A 47 49 56 45 20 42 55 46
3F20 46 45 52 20 52 41 4E 47 45 3A 00 41 3F 4A 52 3F
3F30 46 4C 3D 75 43 3E 90 49 3E 43 44 3D FF 4B 3C 24
3F40 50 3D 73 42 3D 6F 00 C4 84 90 02 C4 90 CA F1 C2
3F50 F9 CA F3 AA F3 C4 3C 35 C4 E2 31 3D AA F9 90 EF
```

Use this complete hexadecimal listing of the program if you wish to prepare a paper tape or cassette to run the editor in RAM, or if you are able to burn your own PROMs.

Once the text is in the buffer, you can edit it using the commands shown in the table. Note that the L, C, I, D, P and B commands may all have arguments, to specify individual lines or a group of lines. In fact the I command must have one argument, to indicate where the insertion is to take place.

The argument number or numbers must precede the command letter. Thus to list lines 12 to 15, for example, you simply type 12, 15L followed by a carriage return.

To change a line, say line 34, you simply type 34C, a carriage return, and then type in the new line text. Similarly to insert a new line or lines before an existing line, say line 17, type 17I followed by a carriage return and then type in the extra lines. To delete lines, say lines 20, 21 and 22, type 20, 22D and then a carriage return.

A single argument implies that the command should affect only the one line. Two arguments imply that the command should affect all lines between the two corresponding lines, inclusively. Thus 15, 20C implies that six new lines are to be fed in, replacing the existing lines 15, 16, 17, 18, 19 and 20.

Note that following the input of an A, R, or I command, the editor enters its text-input mode of operation. It will normally remain in this mode until it detects a "bell" character, whereupon it returns to the command mode, ringing the bell to indicate its response. If a tape being read in following the R command is terminated with a bell character (having been punched previously using the B command), this will take place automatically at the end of the tape.

All commands other that A, R or I cause the editor to return to command code automatically when the command function has been performed.

Note that the editor will also return to the command mode automatically during the text input (including during editing which involves additional text), if either of the internal line or text buffers become full.

The number of text lines held in the editor's buffer may be found at any time by typing an oblique (/) while in command mode. The editor will immediately type the current line total, in decimal.

When accepting text from the keyboard, the editor will detect a "percent" character (%) and interpret this as a destructive backspace. This may be used to correct character errors if they are noticed very shortly after entry (i.e., before the end of the line). Each percent character is interpreted as a one-character backspace, and the character may be used repeatedly to backspace right back to the start of the current line, if necessary.

Before punching a tape following the P or B commands, the editor pauses to allow the user to turn on the tape punch. Typing any character on the keyboard then causes it to punch a leader, the desired output text (followed by a bell character in the case of the B command), and finally a trailer. It then pauses once more, to allow the user to turn off the tape punch. After doing so, the user must then signal the editor to continue by typing any character via the keyboard.

The editor will automatically query any illegal commands, including commands which are not expressed in the correct form. It will also query you if the arguments in a command are either inappropriate to the command, or not valid for the text currently held in the buffer. For example if you have only 30 lines of text and you give it a command like 45L, it will promptly reply with a query.

The editor has been fairly extensively tested, and to the best of my knowledge it is now free of bugs. However, if you should find a bug, I would appreciate being advised so that all users can be informed.

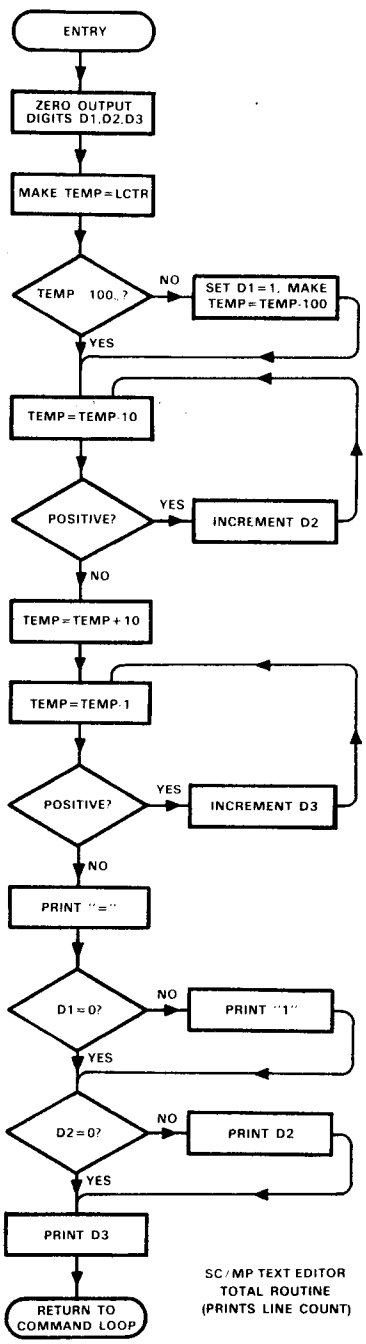
PLEASE NOTE:

Contrary to what is written in the adjacent text, do NOT use a space character to separate the start and finish hex numbers keyed in to define the available memory buffer range, when the Editor requests this upon start-up. A space is ignored by the software, causing a malfunction. You can use almost any other non-hex character — any of the punctuation marks, or any of the alphabetic characters from G to Z inclusive.

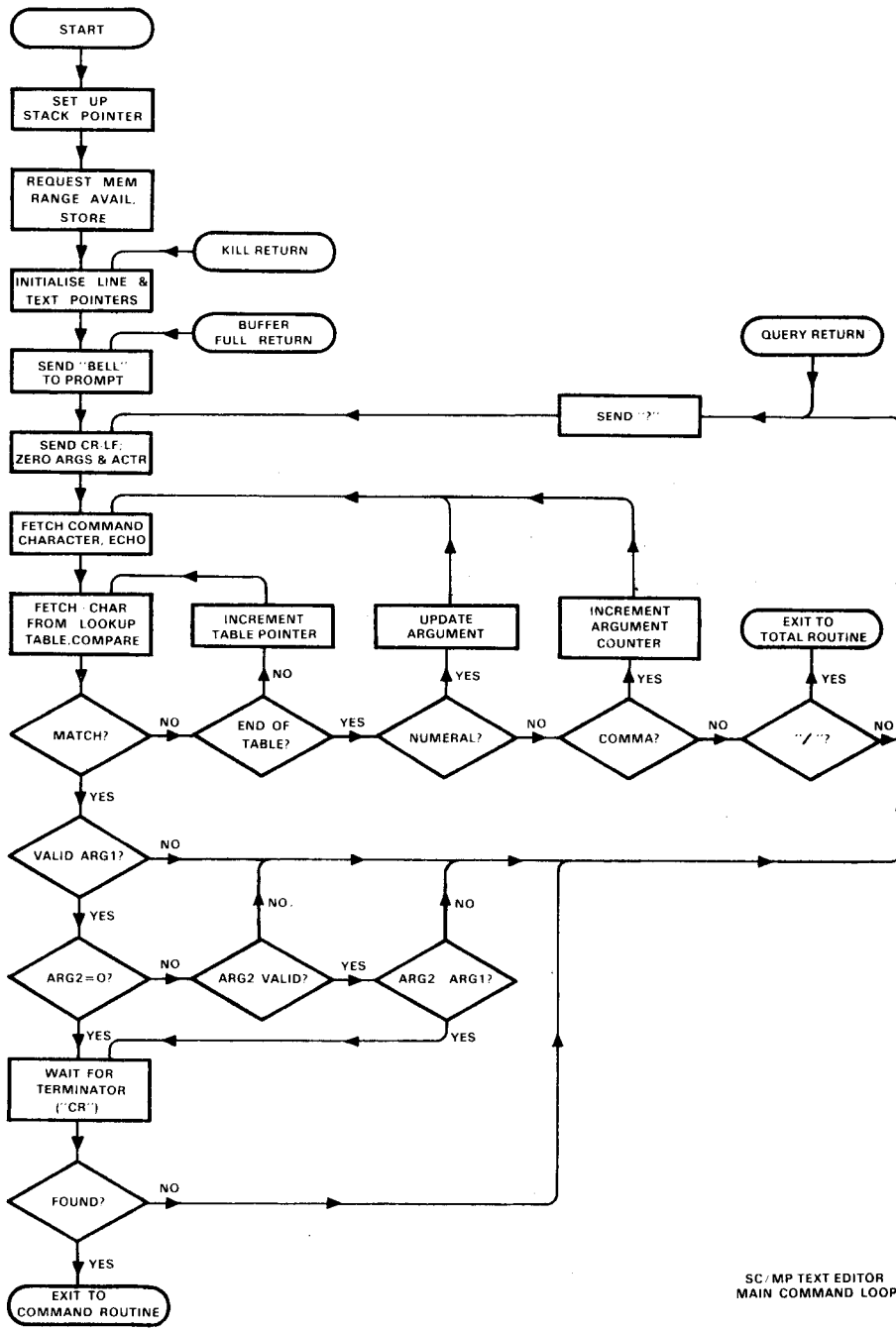
Flowcharts for the SC/MP

by JAMIESON ROWE
and DANIEL HOOPER

Space restrictions prevent us from giving a full listing of the text editor program described in the preceding article. However to give interested readers an idea of how it performs the various functions, here are the detailed flow charts. These should also help those wishing to write a similar program for systems using other microprocessors.

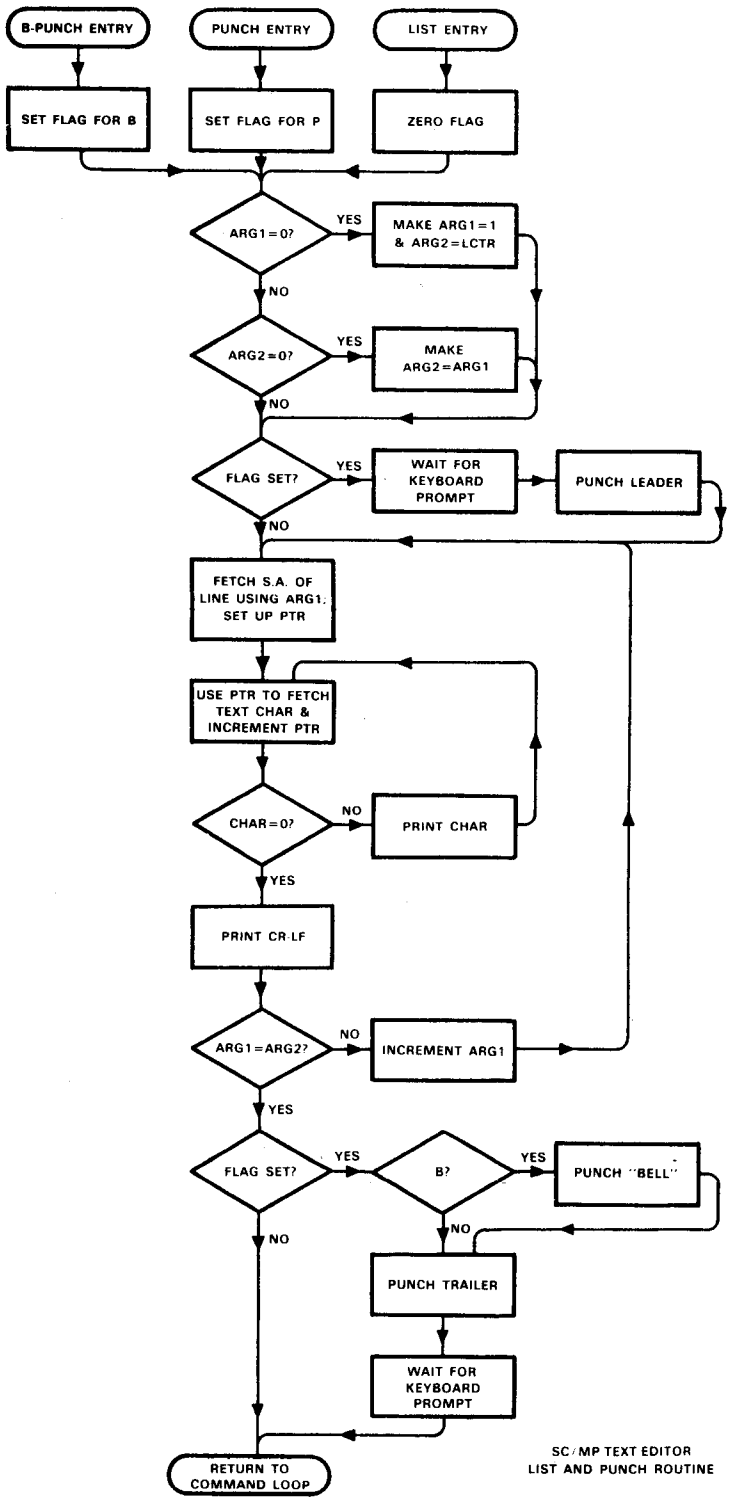


SC/MP TEXT EDITOR
TOTAL ROUTINE
(PRINTS LINE COUNT)

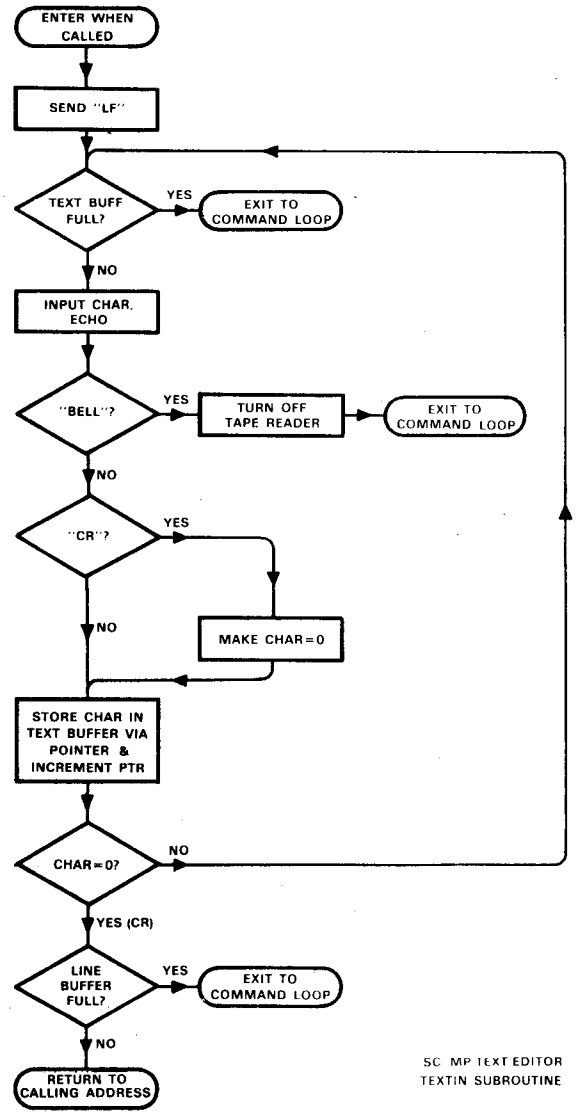


SC/MP TEXT EDITOR
MAIN COMMAND LOOP

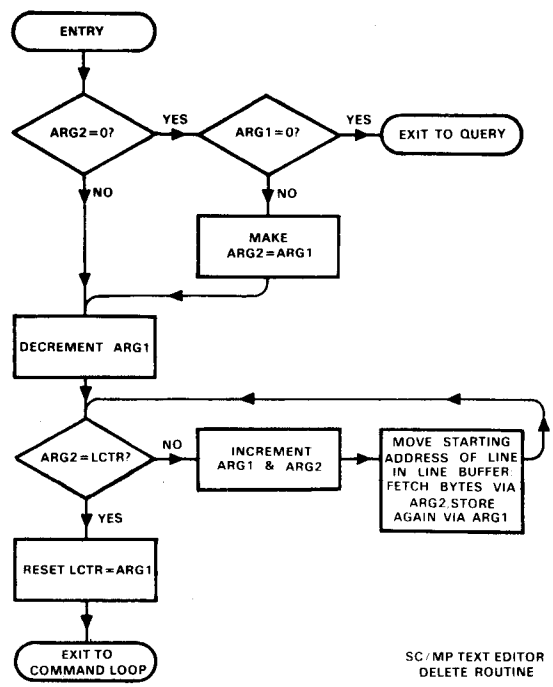
text editor



SC/MP TEXT EDITOR
LIST AND PUNCH ROUTINE



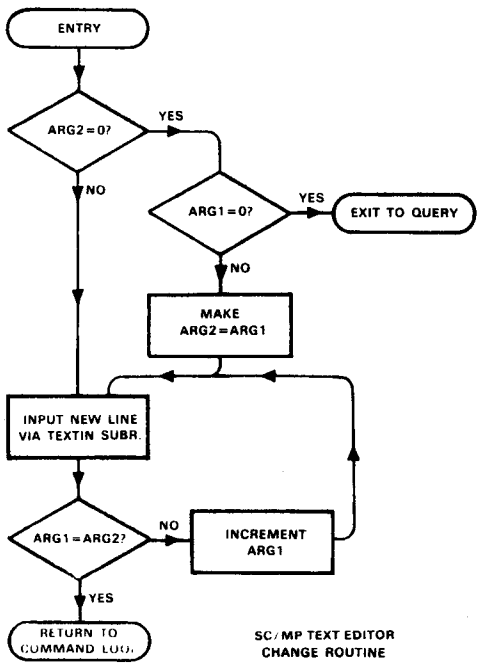
SC/MP TEXT EDITOR
TEXTIN SUBROUTINE



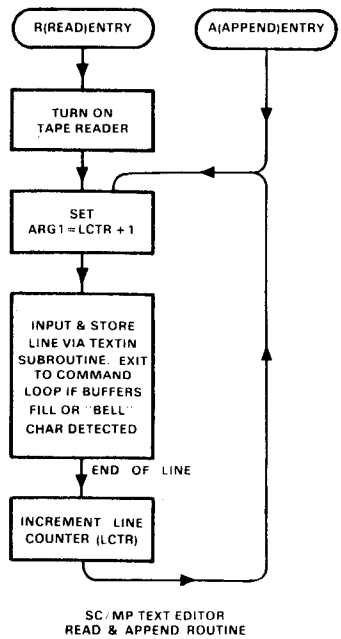
SC/MP TEXT EDITOR
DELETE ROUTINE

Flowcharts for the SC/MP editor. . .

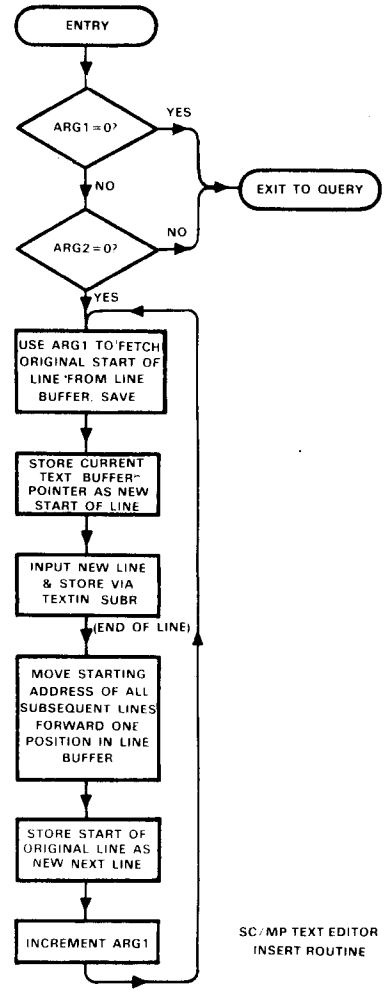
Here are the three remaining sections of the SC/MP text editor program, in flowchart form. The routine at lower left is that for changing existing lines; that at lower right performs the READ and APPEND functions, while that at right is for inserting additional lines after a designated existing line in the text.



SC/MP TEXT EDITOR CHANGE ROUTINE



SC/MP TEXT EDITOR READ & APPEND ROUTINE



SC/MP TEXT EDITOR INSERT ROUTINE

Assemble your own Mini Scamp programs!

Once you have built the Mini Scamp, the next job is to write some programs and run them. In most cases this will have to be done the long way, using hand coding and entry. For those who haven't done this before, the following article will show you how.

by PETER LAZARUS*

Simple microcomputers like Mini Scamp can only run programs which are in the form of so-called "machine language". This is really nothing more than a string of 8-bit binary numbers, which are stored in consecutive memory locations. Most of the 8-bit numbers are code numbers representing particular instructions in the microprocessor's repertoire.

It is not all that easy for we humans to visualise a program in its machine language form, however, so programs are actually written in what is called a "symbolic language". The very simplest type of symbolic language is where two-digit hexadecimal numbers are used to represent the 8-bit machine language code numbers; as you might expect this is only slightly more convenient than true machine language.

A more convenient type of symbolic language is one where short easily remembered mnemonic words are used to represent each type of machine instruction. Words like "STR" to represent a store instruction, "LD" to represent a load instruction, and "JMP" to represent a jump instruction. A program written in this sort of symbolic language is very much easier to visualise and follow from a human point of view.

Of course after a program has been written and checked in a mnemonic language of this type, it must still be translated into the equivalent machine language code understood by the computer. This translation into machine language is known as "assembly".

With larger computers this assembly can be done by the computer itself, under the control of a specially-written "Assembler" program. Or it can be done by another computer altogether, under the control of a suitable "Cross Assembler" program.

For those with access to a larger computer, assembling programs can thus be quite easy. But for those of us who haven't got access to a larger machine, the task has to be done the long way. This article aims to show you how to do this for yourself, for Mini Scamp and other small microcomputers based on SC/MP.

First of all, you will need to write your program in mnemonic language. If you haven't done this before, I recommend that you get the SC/MP Programming and Assembler Manual published by National Semiconductor (Publication Number 4200094B). It costs around $10, but is virtually essential for serious programming—particularly if you haven't any previous experience.

The lower cost SC/MP Technical Description (Publication Number 4200079A) has some information on instruction formats, and an instruction summary, but this isn't really enough unless you have a fair amount of previous programming experience.

Both publications are available from National distributors, and also from suppliers like Dick Smith Electronics and Radio Despatch Service.

After writing a program, and before attempting to assemble it into machine language, you should take a sheet of paper and execute the program yourself, pretending to be the microprocessor. Follow each instruction in the program literally, writing the result at each step. In this way you should find any errors, and be able to correct them. It is important to remove as many mistakes as you can at this stage, as correcting them later can involve much more effort.

Now we are ready to assemble the program into machine language. To do this the operation code or "op-code" for each instruction must be found, together with the number of 8-bit words or "bytes" involved for each instruction.

To serve as an example, I will use the simple program given in the April Mini Scamp article. This is reproduced in Fig.1.

A worksheet should now be drawn on a piece of paper. The program should be copied onto it in the space on the right, labelled source code. The worksheet format is shown in Fig.2. For convenience a list of the particular SC/MP instructions used in the sample program, their opcodes and formats are shown in Fig.3.

The first task is to write the opcodes of the instructions in the Code column. At the same time the length column of the worksheet can be filled.

The first instruction is NOP. From Fig.3, we determine that the opcode for this is X'08 (meaning 08 hexadecimal) and the instruction length is 1. So put "08" in the code column, and "1" in the length column.

The second instruction is a load immediate with an opcode of X'C4 and a length of two bytes. So put "C4" in the code column, and "2" in the length column.

So far the process has been simple. The next, Exchange Pointer High (XPAH) instruction has a basic opcode of 34 and

```
        *BINARY COUNT AND DISPLAY.

            NOP
            LDI     8
            XPAH    1
            LDI     0
            XPAL    1
LOOP        ST      2(1)
            DLY     255
            ILD     COUNT
            JMP     LOOP
COUNT       .BYTE   0
```

Fig. 1: Dr. Kennewell's counting program used here as a sample for assembly.

*62 Collinson St, Keilor Park, Victoria 3033

a length of one byte. However this one is different to the previous ones. The opcode must be modified to contain the pointer register number. (When 'disp (pointer)' is shown in the Format column, the pointer register number is the one written in brackets.) For all other instructions having 'pointer' in the format shown in Fig.3, the opcode is similarly changed. In the case of Jump if Not Zero (JNZ) for example, the basic opcode is X'9C, but if pointer two is to be used, it becomes X'9E.

The instruction XPAH specifies pointer 1, so its opcode becomes X'35.

We follow the same procedure for all the other instructions in the sample program.

Address calculation is the next step. Now that we know the length of each instruction, we can calculate the starting locations or addresses of our instructions.

Using the SC/MP MPU, our program should begin at address zero, as the SC/MP will automatically start here after reset is pressed.

Place zero in the address column next to the first instruction. Add the first instruction length to this to get the address of the next instruction, and so on. Note that the address must be written in hexadecimal, not decimal. The hexadecimal address can be directly keyed on the Mini Scamp address switches, while decimal values would all have to be converted.

The data can be handled in the same way as the instructions were. Next to each .BYTE we place the value shown on the right. In the example, .BYTE results in a code of 00. For decimal eleven, we would code X'0B (.BYTE 11) i.e., decimal 11 converted to hexadecimal 0B. Alternatively, .BYTE expressions can be written directly in hexadecimal, such as X'7D. And in this case, the code is the same, X'7D. After writing the code for the .BYTE locations, the address can be calculated. Each .BYTE occupies one byte—i.e., a single 8-bit memory location.

The worksheet will now look like Fig. 4. This completes the first stage or "pass" of the assembly.

The second stage is to resolve labels and displacements. Labels are the names given, for convenience, to statements or locations in the mnemonic language version of the program. The example uses two such labels, 'LOOP' and 'COUNT'. Each label appearing in the operand area has to be changed to a hexadecimal code specifying its location in memory.

Displacements when specified in the Format column of Fig.3, usually represent the change of a label to a code defining its location. Displacements can also be explicitly defined, as in the DLY instruction in the example. Here, the 255 is the displacement part of the instruction, used to specify the period of the delay.

Firstly, let's handle explicit displacements. The two LDI instructions have a displacement of 8 and 0. These each occupy one byte, and are written in the code column as 08 and 00. Instructions XPAL and XPAH do not have displacements. The '1' is the pointer register, and that has been handled previously. The 'ST' has a displacement of 02, and finally the 225 displacement in the DLY instruction has to be changed to its hexadecimal equivalent of FF.

To calculate label displacements we must distinguish between two kinds. Firstly there are the labels of data areas referenced by load or store instructions, etc., and secondly there are labels used in Jumps.

Labels used in jump instructions must be changed to a displacement corresponding to the address BEFORE the one actually required. This is because the SC/MP increments its Program Counter just before fetching the next instruction.

The rule for determining the displacement for Jump instructions is:

disp = (address of label) − 1
+ two's complement of
(address of instruction + 1)

For all other label displacements the rule is:

disp = (address of label)
+ two's complement of
(address of instruction + 1)

Although these are expressed in binary notation, it is more convenient to think in hexadecimal. To calculate the two's complement of a hexadecimal number, subtract each digit from X'F, write the result, and then add one. For example, to find the complement of X'25: 15−2=13 (X'D), 15−5=10 (X'A), giving X'DA, then add one giving X'DB.

Now let's calculate the displacement of label 'LOOP'. This is a Jump instruction, so the first rule applies.

Address	Code	Length	Source Code

Fig. 2: (above): Suggested format of a worksheet for hand assembly of programs.

Fig. 3 (right): Op codes and formats for the instructions in the sample program, for reference.

Instruction and Format	Length	Opcode	Description.
DLY displacement	2	8F	Delay
ILD disp(pointer)	2	A8	Memory Increment and Load
JMP disp(pointer)	2	90	Jump
LDI displacement	2	C4	Load Immediate
NOP	1	08	No Operation
ST disp(pointer)	2	C8	Store
XPAH pointer	1	34	Exchange Pointer Low
XPAL pointer	1	30	Exchange Pointer High

Address of label: 0007
Address of instruction + 1: 000E
Two's Complement FFF2
Add: FFF9
Subtract 1: −1
Result (take last two digits): FFF8
So the displacement is X'F8.
For the label 'COUNT' use rule two.
Address of label: 000F
Address of instruction + 1: 000C
Two's Complement FFF4
Add: 0003
Result (take last two digits): 03

Note that in both types of displacement calculation, if the first two digits are anything other than X'00 or X'FF, then the displacement is greater than 127, and with SC/MP a different addressing scheme must be used. Also a displace-

ment of X'80' (−128) is not to be used, as SC/MP will use the extension register for the displacement.

In these examples the pointer register is zero (none was specified in brackets) indicating the Program Counter. If a pointer register is to be used, then the address of the instruction plus one is to be replaced by the address loaded into the pointer register. This applies to both the above rules.

Now our program is fully assembled. It will look like Fig. 5. To enter into the memory, we consider only the address and code parts of the worksheet. Entering programs into the Mini Scamp was described in Dr. Kennewell's first article, in the April issue.

Lastly, a word about program alterations. If you want to change an instruction, it can be done provided the new instruction length is equal to or shorter than the original. If the length is equal, then change the hexadecimal code to reflect the new instruction. If the length is shorter, the new instruction occupies the first byte and a NOP (X'08) can be used to occupy the second byte.

To add extra instructions in the middle of a program can involve a lot of work—you have to re-assemble the whole program again! The easy way is to add them at the end of the program, and provide a Jump at the point you want them executed. Say for example we wanted to add three extra instructions after label 'LOOP' in our example. The three extra instructions would be added at the end (address X'0010). The DLY instruction can be replaced by a JMP instruction to transfer control to address X'0010.

Address	Code	Length	Source Code		
0000	08	1		NOP	
0001	C4	2		LDI	8
0003	35	1		XPAH	1
0004	C4	2		LDI	0
0006	31	1	LOOP	XPAL	1
0007	C9	2		ST	2(1)
0009	8F	2		DLY	255
000B	A8	2		ILD	COUNT
000D	90	2		JMP	LOOP
000F	00	1		BYTE	0

Fig. 4: How the worksheet for the sample program should look after op codes, lengths and addresses have been added to the source code.

We have to add the DLY back again before the three new instructions, and add another JMP at the end to return to X'000B. The program would look like:

```
LOOP    ST              2(1)
        JMP             EXTRA
RETN    IDL             COUNT
        .
        .
COUNT   .BYTE           0
EXTRA   DLY             255
        Extra instructions (3)
        JMP             RETN
```

This technique can be particularly handy for temporary repairs to a program, to get it going. Whether you leave the "patches" in permanently, or rewrite the program later to make it more elegant, is up to you.

Now it's your turn to write and assemble some programs. Try simple programs of no more than 20-30 statements at first, as you could quickly get discouraged attempting larger ones initially. You might attempt assembly of the other example given in the April issue, to check your understanding.

Address	Code	Length	Source Code		
0000	08	1		NOP	
0001	C4 08	2		LDI	8
0003	35	1		XPAH	1
0004	C4 00	2		LDI	0
0006	31	1		XPAL	1
0007	C9 02	2	LOOP	ST	2(1)
0009	8F FF	2		DLY	255
000B	A8 03	2		ILD	COUNT
000D	90 F8	2		JMP	LOOP
000F	00	1	COUNT	.BYTE	0

Fig. 5: The worksheet for the sample program when fully assembled, with all displacements added in the code column.

Resident Assembler for the SC/MP LCDS

National Semiconductors has announced the release of a line-by-line resident assembler for the SC/MP Low Cost Development System (LCDS). Known as SUPAK, the assembler comes in eight 512-byte PROMs or ROMs, which plug into a standard ROM/PROM card.

In the 4k-byte firmware package are actually three programs: a line-by-line assembler, a paper tape line editor and a PROM tape punch program.

The line assembler accepts a program written in limited SC/MP assembly language from a keyboard or paper reader, and assembles it directly into RAM. The editor allows insertion, deletion or replacement of lines in program source code, while the PROM tape punch will punch out a selected part of RAM for PROM programmers such as the DATA I/O, in appropriate format.

Priced at $300, SUPAK will be available shortly from NS distributors.

Using Mini Scamp to generate random numbers

Here is another article by the original designer of the Mini Scamp project, this time to help you become proficient at programming. It explains how a computer may be used to generate random and pseudo-random numbers, and gives a Mini Scamp program which demonstrates pseudorandom number generation.

by DR JOHN KENNEWELL
Physics Department, Newcastle University

A program that generates a sequence of random numbers finds many applications in the world of computing. Such programs were used in one of the first electronic computers ever constructed, at the Los Alamos laboratories during the second world war. Here, a technique known as 'Monte Carlo simulation' employed random numbers to calculate the critical masses of uranium or plutonium needed for a nuclear explosive device.

Many games programs employ random number generators. These enable a computer to simulate the tossing of a coin, the throwing of dice, or the choosing of a card. To a certain extent random number generators can be used to make a computer appear more intelligent, or perhaps we should say more human. This is achieved by having not one, but a list of possible responses to any given situation, and then using a random number to decide which of these responses will be actually given at any particular time.

In larger computers most random number programs use the mathematical expression

$$R_{n+1} = (P \times R_n) \text{ modulo } Q$$

to generate a sequence of random numbers. R_n is the last random number calculated, and this is used to produce the next random number R_{n+1}. To start with, R_n can be put equal to any number, and this number is termed the 'seed', from which all later numbers will 'grow'. P is a suitable prime number and Q is usually 2^N where N is the number of bits per word in the computer (e.g., N = 8 for Mini Scamp). The expression 'modulo' means that the product $P \times R_n$ is divided by Q and only the REMAINDER is retained. For example 9modulo5 = 4 or 7modulo3 = 1. This type of modulo division is done very simply if $Q = 2^N$ by simply ignoring the fact that overflow has occurred in the multiplication of $P \times R_n$.

The above procedure, while quite simple, and not impossible to program for Mini Scamp, does require a multiplication routine, which could use up a considerable number of memory locations. An alternative random number generator can be obtained by simulating a shift register with feedback (see Fig. 1). Starting with any number except zero in the shift register, the next number in the sequence is obtained by shifting all bits in the register one place to the right. Bit 7 will be lost, and bit 1 is formed by EXCLUSIVE-ORing bits 6 and 7 (termed the feedback bits) of the initial number. Following numbers in the sequence are generated by simply repeating the above procedure.

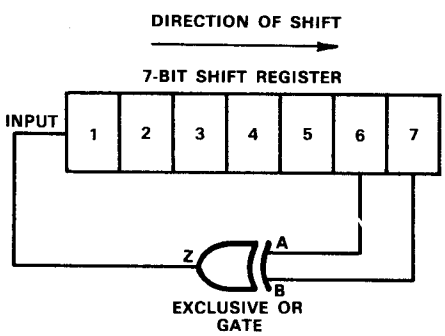


Fig. 1: A 7-bit pseudorandom sequence generator as implemented in hardware.

In point of fact the above method does not produce truly 'random' numbers. It would be more correct to say that it generates a pseudorandom sequence of numbers, because given any starting number, the sequence will always be fixed. In the case of a 7-bit register, the sequence will consist of 127 different numbers before it then starts to repeat.

It turns out however, that such a pseudorandom sequence is more useful in programming than a truly random sequence, particularly when first testing a program. Imagine testing a program that uses totally RANDOM numbers. Every time you ran it, you would obtain different results. Under these conditions it would be very difficult to tell whether the differences were due simply to the random numbers, or to a fault in your program. With a pseudorandom sequence you know that, as long as you start with the same 'seed' number each time, the results will be repeatable.

As long as the sequence is made large enough, a limited number of values from the sequence will always appear random. The randomness of such a sequence can also be increased by considering only a smaller number of bits than are used in the shift register (e.g. the lower 8 bits of a 15-bit register).

A program to implement the above-mentioned procedure is given here for the Mini Scamp. The accumulator is used as the shift register. Excluding the NOP instruction, the next four instructions are concerned with loading pointer register one with the base address for the LED's (X'0800). In this way, each random number generated can be displayed on the front panel. The next instruction loads the accumulator with the seed number which has been planted in location 'X0028, and as long as this is not zero the

INPUT A	INPUT B	OUTPUT Z
0	0	0
0	1	1
1	0	1
1	1	0

Here is the truth table definition of the exclusive-OR function, for reference.

Shift Register Length (bits)	Feedback Bits	Sequence Length
3	2,3	7
4	3,4	15
5	3,5	31
6	5,6	63
7	6,7	127
8	2,3,4,8	255
9	5,9	511
10	7,10	1023
11	9,11	2047
12	2,10,11,12	4095
13	1,11,12,13	8191
14	2,12,13,14	16383
15	14,15	32767
16	11,13,14,16	65535

Fig. 2 (left): how to make pseudorandom generators using shift registers of different lengths.

At right is the listing for the author's program to duplicate the function of the circuit of Fig. 1.

```
                        *RANDOM NUMBER GENERATOR
0000 08                 NOP
0001 C400               LDI   0
0003 31                 XPAL  1
0004 C408               LDI   8
0006 35                 XPAH  1
0007 C020   RANDOM      LD    RND
0009 9C02               JNZ   START
000B C401               LDI   1
000D D403   START       ANI   X'03
000F 9808               JZ    ZERO
0011 FC02               CAI   2
0013 9404               JP    ZERO
0015 C480               LDI   X'80
0017 9002               JMP   FIN
0019 C400   ZERO        LDI   0
001B D80C   FIN         OR    RND
001D 1C                 SR
001E C809               ST    RND
0020 C902               ST    2(1)
0022 8FFF               DLY   255
0024 8FFF               DLY   255
0026 90DF               JMP   RANDOM
0028   43   RND          BYTE  67
```

bit to be fed back to bit 1 can then be computed. If however, the seed is zero, a 'lock-out' condition will occur, as the EXCLUSIVE-OR of 0 and 0 is always 0. To avoid this possibility a value of 1 is thus placed in the accumulator.

The ANI X'03 instruction is now used to mask off all bits except 6 and 7, the lowest significant bits, as these are the feedback bits. Note that the most significant bit of the accumulator is ignored as far as the random numbers are concerned, leaving the 7 lower bits as required.

After the mask has been applied, the accumulator may contain any number from 0 to 3 inclusive. If it is zero, then the new bit one will be zero (JZ ZERO). If 2 is subtracted from the accumulator (CAI 2) and the result positive (initial number would have been 3 in this case) then the bit fed back will again be zero (JP ZERO). If neither of these conditions is true, then either bit 6 or bit 7, but not both, would have been one, and the bit to be fed back should be one. This bit is loaded temporarily into the most significant bit of the accumulator (either by LDI X'80 or LDI 0, whichever is appropriate) OR'ed with the original random number (OR RND) and then shifted right (SR) one place.

The newly created number then replaces the old random number in location X'28 and is also displayed on the LED's (ST 2(1)). The delay instructions slow the sequence down sufficiently for you to observe what is happening.

If you wish the program to stop after each new number generated, then replace the second delay instruction at location X'24 by the instruction LD 1(1) which has the hex code C101. This will cause the machine to hold with the DRQ light turned on until you press the deposit button. What you deposit into the accumulator is unimportant, as it is ignored by this program. Using this method you can either write out the complete sequence, or simply determine mentally what the next number in the sequence should be, and then press deposit to test your prediction.

You will probably note that if you keep your finger on the deposit button, the machine as originally described will continue to sequence as before (actually faster, since one delay has been omitted). Thus, if you only want one number at a time, you must depress and release the deposit button quickly (in less than one-quarter of a second). The reason for this behaviour is that, with deposit activated, the mono in the circuit will be continuously enabled, and if the SC/MP CPU requests a new data value (as it will each time around the program loop), it will have one provided it immediately without having to go into the hold state. This won't happen however, if you have modified your Mini Scamp as shown in the July issue.

It is possible to write a similar program to simulate shift registers of any other length desired. Fig. 2 shows the appropriate feedback bits to use, and the length of the sequence generated for other length registers. After becoming familiar with the program presented here, you might like to try your hand at a program for a longer sequence.

A register length of 15 is quite easy to implement using two 8-bit words. The word containing the lower 8 significant bits is subject to exactly the same test as the for the 7-bit register. However, the bit to be fed back is placed in the MSB of the word containing the higher significant bits. Then after clearing the carry/link (CCL), a rotate right with link (RRL) is performed on this word, causing the LSB of the word to be shifted into the carry/link register. If now a shift right with link (SRL) is performed on the other word, this bit will be shifted into this lower word. The procedure is illustrated in Fig. 3. In this way it is possible to simulate a register of any length whatever.

In actual use, a program such as this would be written as a subroutine in part of a larger program. Each time the main body of the program called the subroutine, it would calculate a new random number for use by that program. In many cases, only a yes/no type decision may be required, and thus only one particular bit of the random number need be considered.

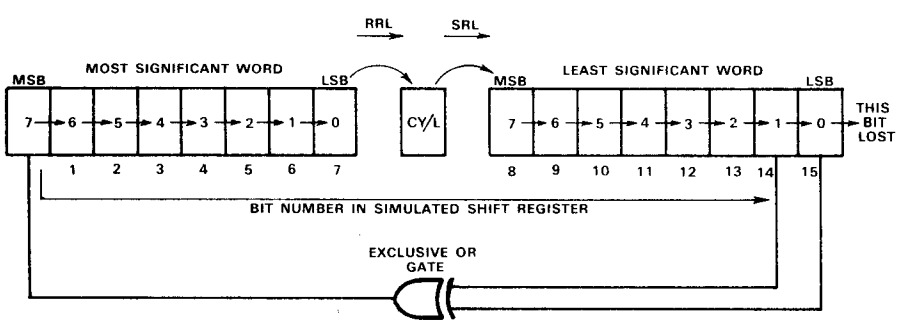


Fig. 3: Suggested way of simulating a 15-bit sequence generator.

A hexadecimal keyboard

Designed to eliminate the tedium of binary input switches, this low-cost hexadecimal keyboard should find immediate appeal with microcomputer enthusiasts who cannot afford a teletype or VDU. The unit uses readily available parts, and can be built around a suitably modified calculator keyboard.

by A. K. LOVEJOY

The project to be described is the result of a personal desire to "tread the middle ground" between an expensive teletype or video display terminal and a cheap but rather tedious binary switch system. By good fortune, the final cost of this project was no more than for a basic switch system—about $10.00.

The unit comprises a "modified" ex-calculator keyboard (minimum 18 keys), a hexadecimal diode encoder, two quad latches and an automatic steering and latching circuit. The output is in the form of two parallel binary-encoded hexadecimal digits (the first we call the left hand or LH and the second the right hand or RH), which by user's choice may be high state or low state logic. The first entry steers left, while the second and subsequent entries steer right.

Keyboard operation commences by pressing the "clear" button to give an output response of 00. The first choice of digit (0-9 or A-F) is now made, and this choice is duplicated on both outputs. However, only the LH digit is "hard latched"; subsequent operations change only the RH digit until the keyboard is cleared. This latter feature simplifies operation, as many operation codes have a common first digit.

The CR (carriage return) command was added to the circuit almost as a postscript, although admittedly this is a desirable feature. Operation of the CR key gives an instant 0D (hex) on the output bus lines, and no clearing is necessary prior to giving the command.

Construction will depend partly upon the choice of keyboard, and selection preference should be in favour of older and larger types which permit easy modification. Alternatively, one could use separate SPST momentary contact pushbuttons.

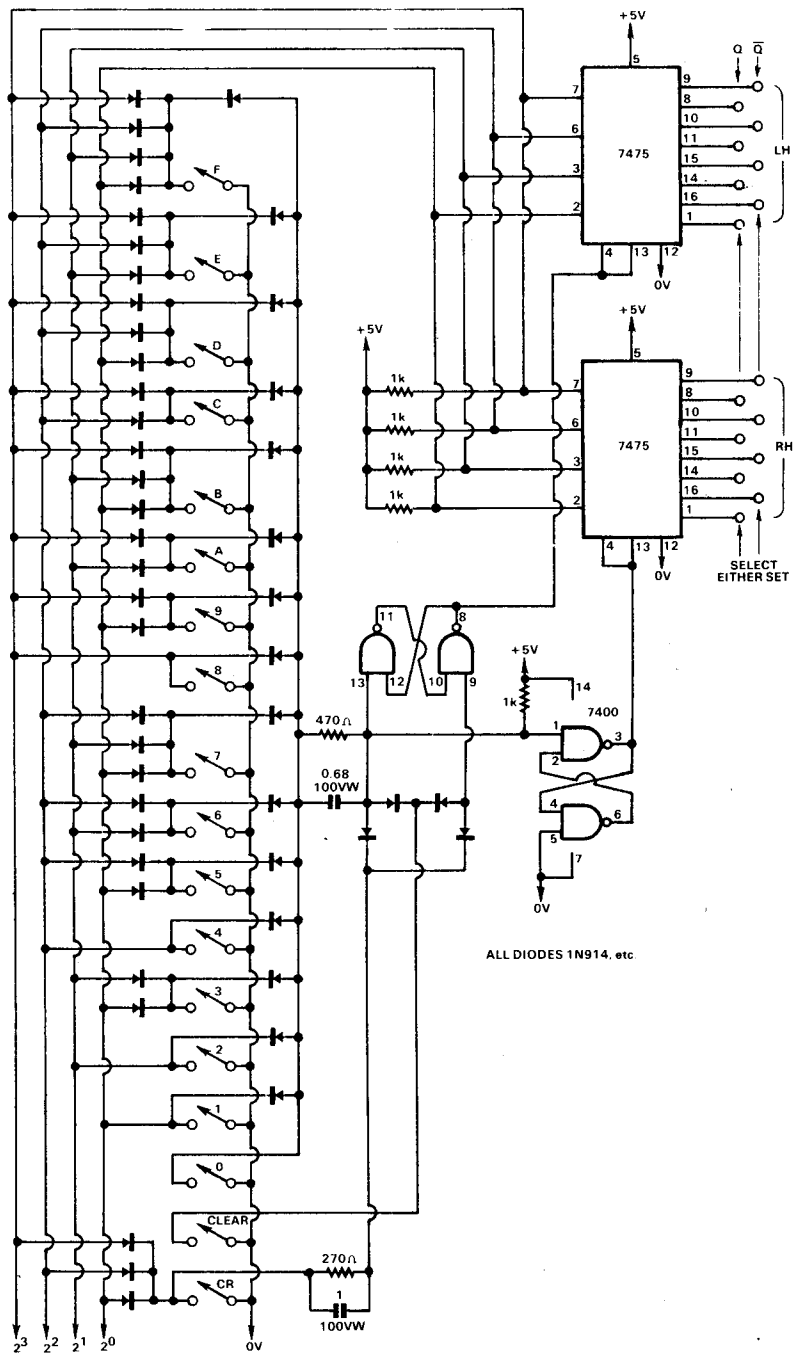
A word of warning concerning the small silicon diodes used in the circuit. Be wary of bulk purchases of so-called "hobby" or "untested" packs. My pack of 50 contained no less than 43 beautifully-encased short circuits!

The prototype was constructed on 0.1in Veroboard and connected to the microprocessor by a short length of 12-wire rainbow cable. The latch output set (Q or Q-bar) is chosen according to whether a high or low state transition is required.

As shown the keyboard is intended for parallel interfacing, in place of a set of binary switches. For serial interfacing, the latch outputs could be connected to the transmitter section of a UART.

Power supply requirements are a modest 75mA at 5V. In most cases, this may be derived from the existing microprocessor supply.

Finally, six of the non-numerical keys have to be re-labelled A-F. The easiest short-term method is to use plastic embossing tape cut into squares and stuck to the keys tops.



HEXADECIMAL KEYBOARD

2/CC/-

National Semiconductor

PACE Low Cost Development System

- > Simplified Interface and Machine Code Development Tool
- > Excellent Training and Learning Equipment
- > Integral Hex Display of Function and Address/Data
- > Integral 20-key Keypad
- > Handles a Variety of Terminals and Serial Peripherals

The PACE Low Cost Development System from National Semiconductor is a fast and efficient vehicle for developing a working microprocessor application system. PACE LCDS features an ease of hardware and software access that facilitates interface and machine code development.

The system contains its own 20-key keypad. Using this, the designer can directly enter data, instructions, and control commands. And on the integral 6-digit hex LED display the designer can easily examine register and memory contents.

For versatility, the LCDS provides strap selectable baud rates and either RS232 or 20 mA current loop serial interfaces. These allow use of a variety of common input/output peripherals, including teletypewriters, CRT displays, and other keyboard devices.

The designer builds application routines or subroutines by entering code directly through a keyboard. Alternatively, he may use a teletypewriter to load paper tape in assembled format. Either way, he has complete flexibility in developing his programs. Using PACE LCDS, he may view, print, and modify memory and register contents. To simplify program checkout and debugging, both single-step and continuous mode with set breakpoints are available.

Once programs are debugged (LCDS contains a powerful debug monitor), the designer outputs the program on punched tape in a format compatible with National's PROM programmers.

Hardware interfaces may be quickly evaluated and debugged by plugging them into the motherboard and exercising them with the CPU. Three on-board connectors accept standard cards to further extend the RAM, ROM and I/O capabilities. PACE LCDS features 60k x 16-bit addressability, with optional split base page operation.

The heart of PACE LCDS is the PACE CPU chip (IPC-16A/520D), supported by the System Timing Element (DP8302J) and Bidirectional Transceiver Elements (DP8300N). LCDS also includes 1k x 16 bits of random access memory, 1k x 16 bits of read-only memory, plus sockets for 1k x 16 bits of user-programmable read-only memory.

The PACE LCDS comes fully assembled and tested on a heavy duty printed circuit board mounted on a 10" x 12" x 3" aluminum chassis.

PACE LCDS: IPC-16P/301, Price: $585.

NS ELECTRONICS —

For further information contact
N.S. Electronics on
Melbourne (03) 729 6333;
Sydney (02) 93 0481;
Adelaide (08) 46 3929;
Perth (092) 25 5722;
Brisbane (07) 36 5061;
Hobart (002) 44 1337

There's more to Microprocessors than just buying chips

Microcomputers, one of the most exciting technological developments of this decade, are being used in hundreds of applications from simple controllers to complex data processing systems. To enable users to efficiently bring microcomputers into their applications, Warburton Franki is offering a selection of workshops that are designed to provide you with the "tools" for making optimum use of Intel microcomputers in system development.

WARBURTON FRANKI

For further information contact the following numbers.
• ADELAIDE 356 7333 • BRISBANE 52-7255 • HOBART 23-1841
• MELBOURNE 699-4999 • PERTH 65-7000 • SYDNEY 648 1711
• AUCKLAND N.Z. 360-154 • WELLINGTON N Z. 698-272

The Microprocessor User Centre