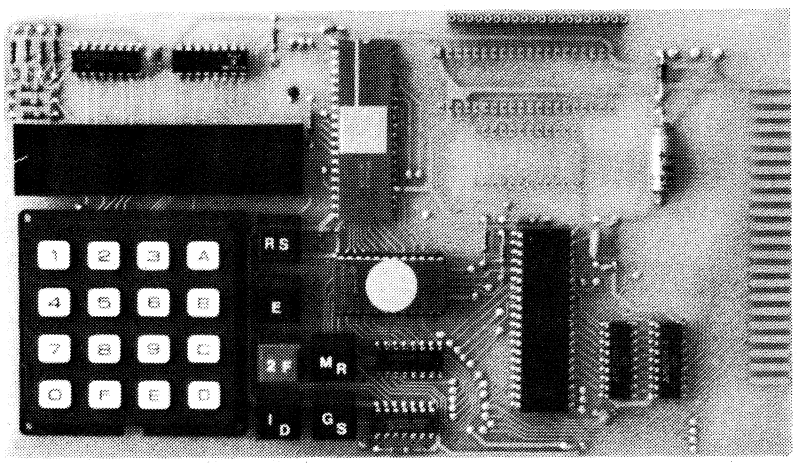


from GAMMATRON "DATUM"

the true starting point

COMPUTER KIT

for $97.00 (inc S.T.).
Postage and handling $3.00
when assembled kit.

READY TO USE

Developed by the S.A. Institute of Technology as a teaching aid for those who wish to learn about microprocessors at minimal cost. The kit uses a MC6802 microprocessor which is complete and ready to assemble with full instructions. Power source is either a reg. 5V Supply or 6V lantern battery provided by the user.

Kit includes P.C. board, all IC's and components, pre-programmed 2716 ROM, key pad.

Send SAE for further information or
send money order, cheque, Bankcard and we will post haste a kit.

For Bankcard users:

Bankcard No.....
Expiry date.....
Name..... Signature .....

Available only from

GAMMATRON

ELECTRONIC SYSTEMS MANUFACTURERS
UNIT 1, WEEN RD, POORAKA, SA 5095
PHONE: (08) 262 6555

DATUM: new low-cost microprocessor trainer

Do you want to learn about microprocessors from the ground up? Have you been following our series on "How to Program in Machine Language" but have no system to test out the concepts you've learnt? Or do you know a bit and are looking for a low-cost dedicated microprocessor system for experiments or for use in a control application? If your answer to any of these questions is "yes" then this is the project for you. Designed initially for teaching microprocessor concepts to students of Electronic Engineering, DATUM is a complete self-contained microprocessor system with everything needed to get you started.

This project got its start when staff at the School of Electronic Engineering at the South Australian Institute of Technology noticed that students with their own microprocessor system at home performed much better than those who had to rely on the limited number of evaluation kits used in the laboratories.

It was obvious that to give every student the same opportunity a low cost microprocessor trainer was required, so the staff set to work. The result was DATUM (Digital Aid for Teaching yoU Microprocessors), an inexpensive, easy to use microprocessor kit based on the Motorola MC6802 processor.

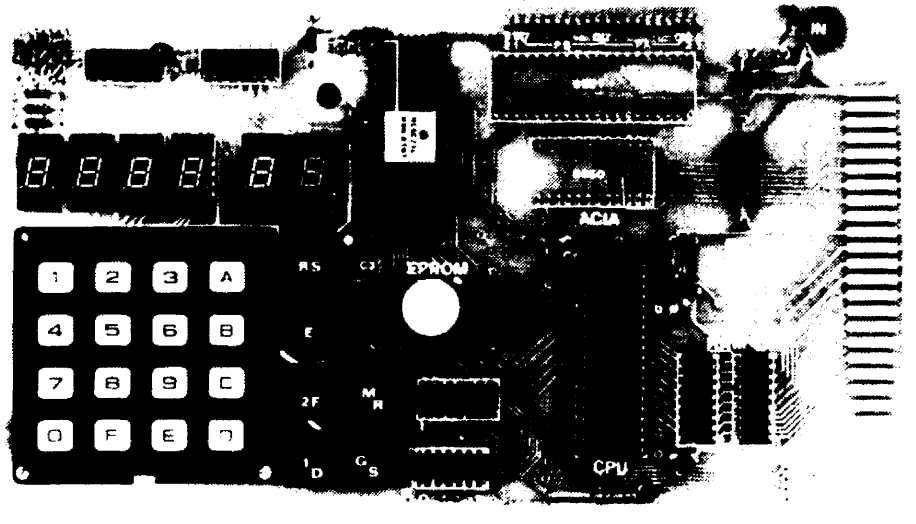
The design and development of DATUM was a co-operative effort, with many of the staff of the School of Electronic Engineering contributing, and a South Australian company, Gammatron, undertaking the production of the printed circuit board and kits for the project. Software for the Monitor program was written by Mr P. D. O'Neill of the Institute.

One of the most important design decisions for the DATUM project was that it had to be a completely self-contained system. A simple keypad on the board provides the means of inputting instructions and data, while output is displayed on seven segment LED displays.

Those readers who have followed our series on "Machine Language Programming for the 6800" will be in their element here, for the only way to program DATUM is by machine lanaguage and the "instruction set" of the 6802 is identical to that of the 6800. Both processors are relatively simple but very versatile, and quite powerful enough for many advanced applications.

There are only two differences between the 6800 and 6802 processors. The 6802 has an on-chip clock generator, needing only an external crystal. More importantly, the MC6802 has 128 bytes of Random Access Memory on the chip, 32 bytes of which can be placed in a low power "stand-by" mode to preserve important data when the rest of the system is off. This is one reason why we chose the 6802 for use in our Car Computer project.

In fact the Car Computer is a good illustration of what is possible with a small system such as this. Besides its use as a learning aid, DATUM can be put to use



DATUM is a low-cost trainer for learning about microprocessors.

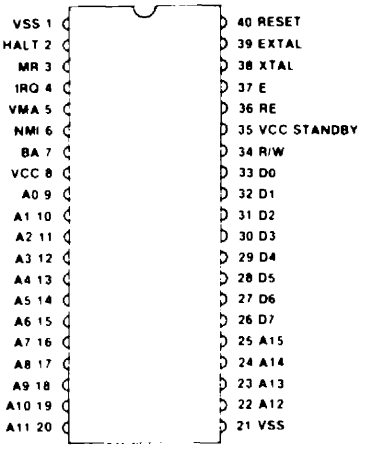


Fig. 1

Fig. 1: Pin-outs of the MC6802.

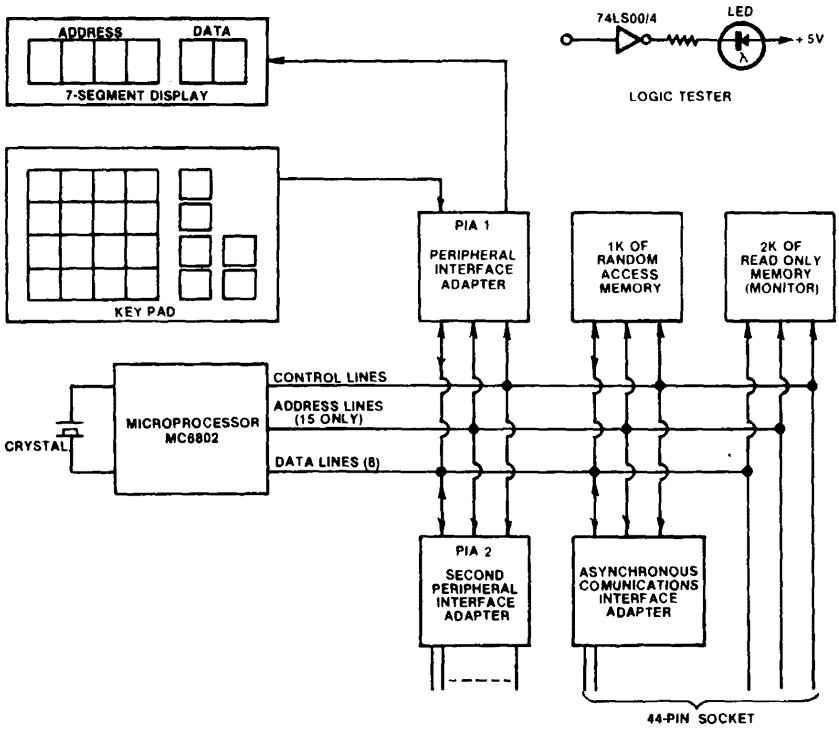


fig. 2: Block diagram of the fully expanded DATUM system.

as a controller around the home (or the factory), or expanded into a larger computer system. Bus signals are brought out to a 44-way edge connector on one end of the board for this purpose and there is provision on the board for additional Input/Output, with a PIA for parallel I/O and an ACIA (Asynchronous Communications Interface Adapter) for serial communication.

The double-sided DATUM printed circuit board measures 228mm x 127mm and is silk-screened with component locations (although not component values). The manual we received had the component values hand-written after the parts list, although it is hoped that this will be corrected in final versions.

The design of a minimum microprocessor system

The heart (or is that the brain?) of any microprocessor system is of course the processor chip itself. Pin connections for the MC6802 are shown in Fig. 1. The processor has 16 address lines, and is thus capable of addressing 2^{16} or 65,536 separate memory locations.

64K of memory is far more than we need, so the first decision made in the design of DATUM was to decode blocks of memory, rather than individual locations. This simplifies the circuitry required and reduces costs, at a negligible penalty – maximum memory expansion. This is 12K bytes in 4K increments.

A minimal system will require both Read Only Memory for permanent storage of an operating program, and RAM (Random Access Memory) for user programs. Static RAM is used since it avoids the complications of refreshing dynamic RAM (see our articles on the Super-80 for details of a dynamic RAM system).

Because we will be entering programs by hand it is unlikely that they will exceed about a hundred instructions. Around 200 bytes of RAM would be sufficient, but the most readily available and cost effective memory for our purposes is the 2114 static RAM chip. This chip is organised as 1K locations, each of four bits, so just two packages will give us 1024 8-bit memory locations.

The ROM package of DATUM holds the Monitor program – a collection of routines which assist the user in communicating with the microprocessor system. Software is provided which scans the keypad, drives the LED displays and accepts input from the user. Depending on this input other routines are activated which allow the contents of memory to be inspected and changed, the internal registers of the 6802 to be displayed and the user's programs carried out.

The Monitor program of DATUM is 1K bytes long, and a complete listing of the source code is included in with the instruction manual supplied with the kit. Since the 2716 EPROM is a 2K device other programs which the user wishes to store permanently can be programmed into the unoccupied addresses. An EPROM programmer suitable for this purpose was described in our January 1982 issue.

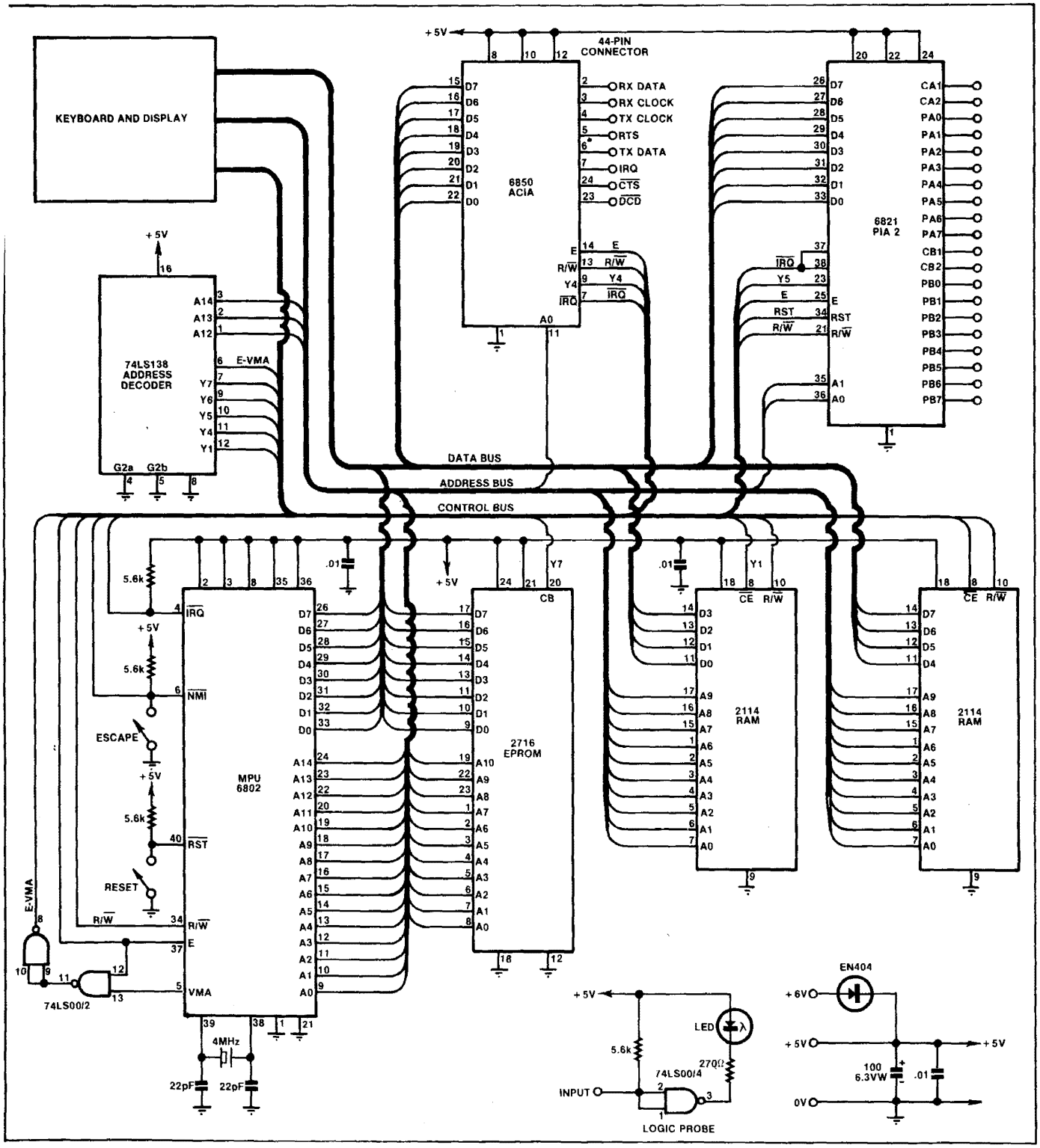
Since DATUM is programmed in assembly language, hexadecimal output was decided on. Each 16-bit address location and 8-bit data byte are considered as combinations of four bits, with each combination represented by one of 16 hexadecimal characters, the numbers from 0 to 9 and letters from A to F. A complete description of the hexadecimal numbering system is given in Part One of "How to Program in Machine Language Language", EA March, 1982.

In hexadecimal, 16-bits are considered as four groups of four, and 8-bit data consists of two groups of four. Six displays are therefore required, four to display the address and two to display the data stored at that address.

A commercially available keypad provides hexadecimal inputs while six individual pushbuttons are used for control functions. Six switches provide the best balance between flexibility and cost. Key functions provided are:

HARDWARE SPECIFICATIONS

Processor: *Motorola MC6802.*
Clock: *1MHz.*
RAM: *1K x 8 on board, expandable to 12K externally, plus 128 bytes on chip.*
ROM: *2K (Monitor program occupies 1K).*
Display: *Seven segment LED readouts (6).*
Input: *Hexadecimal keypad and six function select buttons.*
Output: *Two PIA lines, optional extra PIA, ACIA*
Power requirements: *Regulated +5V or 6V battery at 400mA.*
Programming: *Hexadecimal machine code.*



Circuit diagram of the full DATUM system, less the keyboard and display section shown in Fig. 6.

1. RESET (RS). Initialises all devices and causes the system to jump to the beginning of the Monitor control program.
2. ESCAPE (E). Lets us exit one control function ready for the next control sequence.
3. MEMORY (M). Allows memory contents to be displayed and modified.
4. GO (G). Provides a "start" instruction so programs can be run.
5. REGISTERS (S). Lets us display the contents of the internal registers of the 6802 microprocessor.
6. SINGLE STEP (R). Assists in "debugging" programs by allowing us to step through a program one instruction at a time.
7. INCREMENT/DECREMENT. Lets us inspect memory contents, stepping the address in either direction.

While all these functions were considered essential, some will be used less frequently than others. It is thus possible to share some switches, with functions which are used less often requiring two key presses, the first being the "second function" (2F) key. Other keys such as Reset and Escape must operate directly and cannot be shared. Fig. 3 shows how the seven functions can be accommodated on six switches.

So far we have described a minimum microprocessor system, with a

hexadecimal keypad for input, LED displays for output, a processor and memory. Two additional features have been included on the DATUM board. First, since the system is to be self-contained a simple logic probe has been included on the board so that signals can be monitored. Secondly, provision has been made to expand DATUM when required.

The 44-pin edge connector, visible in photographs of the board, brings out the data, address and control lines for connection to other equipment. In addition there is space on the board for a second PIA (Peripheral Interface Adapter) which provides 20 parallel Input/Output lines and an ACIA, providing a serial interface. With the addition of 12V level translators this chip provides an RS-232C channel for connection to terminals and printers.

With these additions DATUM can become more than a learning aid, finding a host of control and communications functions in real applications.

Fig. 2 summarises the design decisions taken so far and presents a block diagram of the minimum DATUM system.

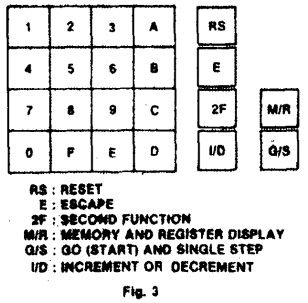


Fig. 3: Alternate key functions are activated by pressing the 2F key then the function key required.

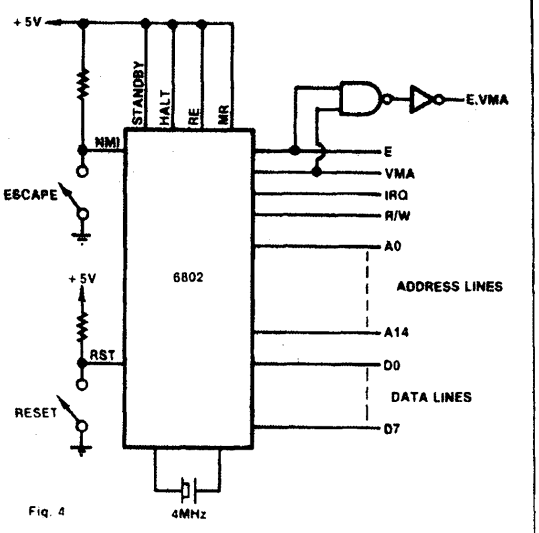


Fig. 4: Data, address and control lines of the MC6802 microprocessor used by the DATUM system.

How it works

Like most microprocessors the MC6802 is "bus" oriented. A common parallel signal path consisting of data, address and control lines connects the main components of the system. Again in the interests of simplicity, not all the control lines of the MC6802 are used in the DATUM design.

The RAM Enable (RE) line is tied to +5V to enable the 128 bytes of on-chip memory. The Memory Ready (MR) line is also tied high. Its normal function is to allow the processor to wait for slow memory devices which are not used in DATUM. "Standby" applies power to the first 32 bytes of the internal memory so that in the event of a power failure this memory can be powered from an emergency supply.

HALT is not used, as we don't want to stop the processor at any time so this line is also tied to +5V. Since HALT is not used, the Bus Available (BA) line which indicates that the processor has halted is not used and can be left open. The remaining control lines are used as follows:

RESET resets the MC6802. A pull-up resistor holds this line high until the Reset pushbutton is operated. NMI, the Non-Maskable Interrupt, is also pulled high and connected to a pushbutton, in this case the "Escape" function. IRQ is the interrupt request line, allowing processing to be suspended and the processor directed to perform another program.

VMA indicates that a "Valid memory address" is available on the address bus, while the R/W line indicates whether a memory operation is a read or a write command. The Enable (E) line is the clock signal for the DATUM system. This clock signal is ANDed with VMA and used to initialise all the units on the bus. The frequency of E is a quarter of the crystal clock frequency, so the 4MHz crystal of DATUM provides a system clock of 1MHz.

Fig. 4 summarises the outputs of the MC6802 microprocessor and the lines used in the DATUM bus system. Pull up resistors have been added to the RESET and NMI lines since these must be held high for the processor to operate correctly.

Address decoding

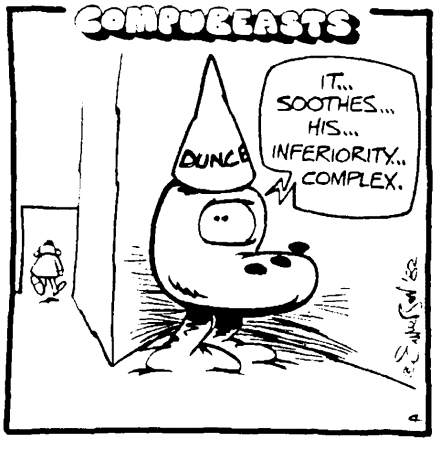
As previously mentioned, not all address lines are decoded by the DATUM circuit. Address line 15 is ignored while A14, A13 and A12 are decoded by a 74LS138 1-of-8 decoder to select individual 4K blocks of memory. The remaining 12 address lines (from A0 to A11) provide a 4K (4096 bytes) addressing capacity. Memory in DATUM is thus divided into eight blocks, each of 4K, for a total addressing capacity of 32K bytes.

The next question is how to allocate the available memory locations. Three constraints are set by the internal organisation of the MC6802 chip:

1. The internal 128 bytes of RAM must be the first 128 memory locations.
2. After a Reset the last processor reads the last two locations in memory (FFFE and FFFF hex) to find the address of the program to be run.
3. After an Interrupt Request (IRQ) the processor reads address locations FFF8 and FFF9 hex to determine the address of the program to jump to.

These last two constraints suggest that the Monitor EPROM should be placed at the top of the "memory map" to permanently store Reset and IRQ pointers at the correct locations. The PIA, ACIA and RAM locations are then allocated in a convenient order as shown in Fig. 5. Note that the PIA and ACIA each occupy an entire 4K block of memory, even though each PIA has only four registers which can be accessed, and the ACIA has only two.

Because of the limited address decoding, each of these registers will be repeated in the memory map 1024 times, but this is not important as long as we are consistent in our programs. Since the amount of memory (RAM or ROM) on the DATUM board is less than 4K, the same memory locations are also repeated throughout the memory map. For example the 2K EPROM is addressed



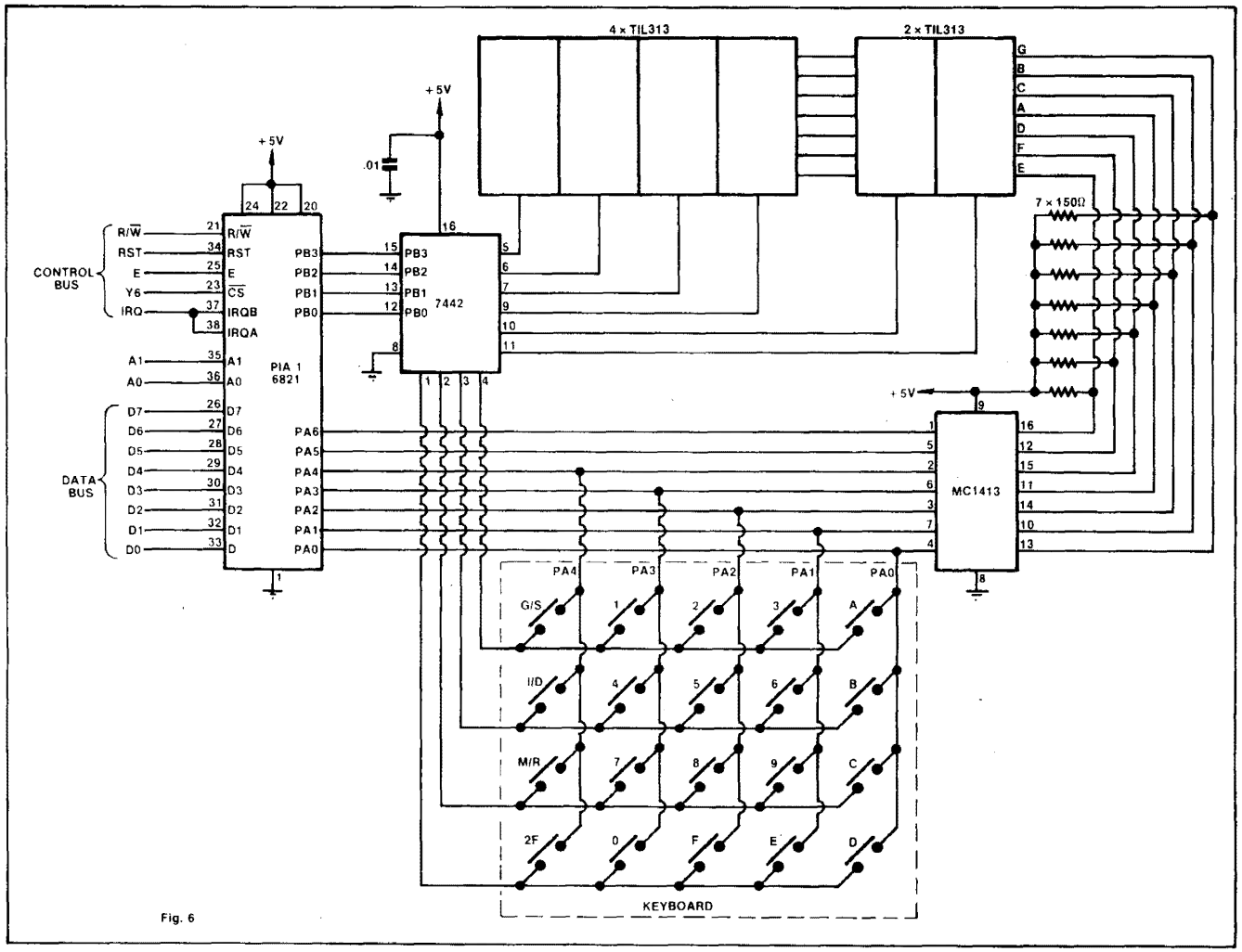


Fig. 6: Circuit diagram of the keyboard and display section of DATUM.

from 7000 (hex) to 77FF and is then duplicated from 7800 to 7FFF hex. Similarly the 1K of RAM is repeated four times, beginning at address location 1000 hex.

This repetition does not matter and any of the appropriate redundant addresses can be used, although a lot of trouble and confusion will be avoided by consistently using the same address locations.

Note that if we ignore the 16th bit of the address bus, 7FFF in hex is the same as FFFF, so the two constraints on EPROM addressing are satisfied even though DATUM has only a 32K addressing capability.

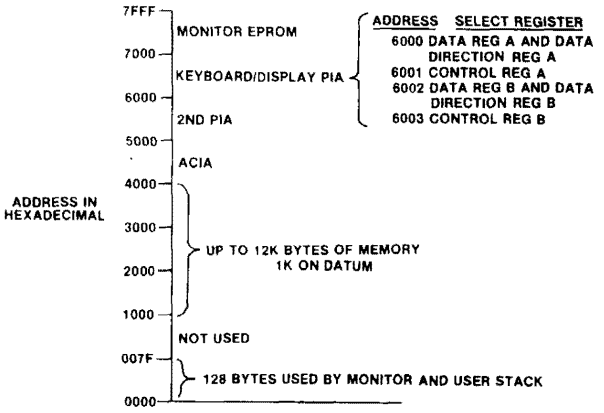


Fig. 5: DATUM memory map shows allocation of ROM, RAM and peripheral adapters.

Keyboard and display

Fig. 6 shows the method of interfacing the keypad and the six displays to the microprocessor. Port B of PIA1 is programmed for output, and four lines, DB0 to DB3, drive a 7442 1-of-10 decoder. Six of the outputs of the 7442 sequentially ground the cathode of each seven segment display while the remaining four lines are input to the rows of the keypad.

A closed key is detected by programming Port A of PIA1 for input and scanning the columns of the keypad through PA0 to PA4. With the four rows of the keypad each driven low in turn a closed key will cause one of the five column lines to go low. The particular key pressed will be at the interesection of the row which is currently low and the column line which is read as low.

For driving the display Port A of PIA1 is programmed for output and seven of the lines (PA0 to PA6) are fed to an inverting buffer to select one of the segments of each seven segment display. All the identical segments of the six displays are connected in parallel and the display illuminated is determined by which

cathode is selected by the outputs of the 7442. The segment driving lines of Port A go low to select a segment, causing the output of the MC1413 inverter to go high and allow the 150Ω resistor to +5V to supply current to the LED segment.

Fig. 7 shows the organisation of the keypad and display circuitry and the way in which each output line is decoded. With this information it is possible to program DATUM to illuminate any segments of the display for a quasi-alphanumeric messages or to monitor a particular key of the keypad.

Note that pins DA7 and DB7 of the PIA1 are not used for the keypad or display and are brought out to pins on the board for experimental purposes. More details of the Peripheral Interface Adapter chip can be found in "How to Program in Machine Language", in EA, August 1982.

Logic probe

One element of the 74LS00 NAND package on the DATUM board drives a LED which serves as a power-on indicator and a simple logic probe for checking wiring and the operation of programs using the PIA outputs.

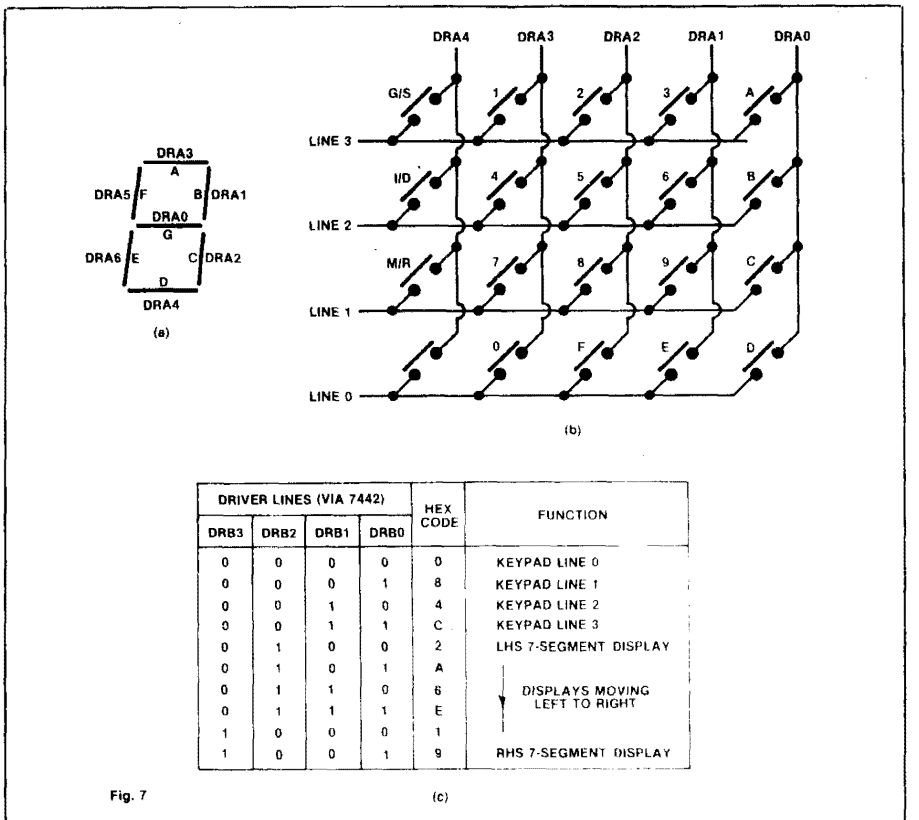
A pull-up resistor ensures that the LED will be on when there is no input to the probe, indicating that power is available to the circuit. This has the unfortunate side effect that if the logic probe input is applied to a pin which is in fact open circuit the LED will be lit, indicating a high state. Despite this drawback the probe is still useful. If for instance, the input of the probe is connected to pin 37 of the microprocessor (the clock output) the LED will be lit if the oscillator is functioning but its intensity will be less than when the probe input is connected to +5V.

The final system

Page 89 shows the circuit diagram for the complete DATUM system, excluding the keyboard and display sections shown in Fig. 6. Fig. 8 shows the connections for the 44-pin edge connector while the printed circuit board overlay is shown at right.

Power to DATUM can be provided either via the edge connector or via pins on the PCB (labelled DC volts in). Power can be supplied from a regulated 5V source or by a 6V lantern battery. Current consumption is around 400mA.

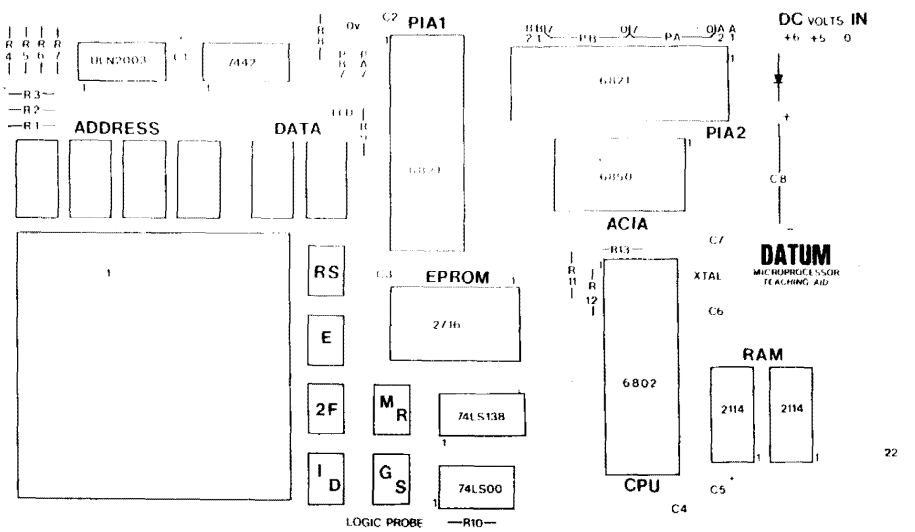
If a battery is used power can only be connected to the pins on the circuit board labelled 0 and +6V which is connected via a series diode. The diode prevents the power being applied with the reverse polarity and drops the 6V to approximately 5.4V. Under no circumstances should 6V and 5V supplies be used at the same time as the diode will be destroyed.



DRIVER LINES (VIA 7442)				HEX CODE	FUNCTION
DRB3	DRB2	DRB1	DRB0		
0	0	0	0	0	KEYPAD LINE 0
0	0	0	1	8	KEYPAD LINE 1
0	0	1	0	4	KEYPAD LINE 2
0	0	1	1	C	KEYPAD LINE 3
0	1	0	0	2	LHS 7-SEGMENT DISPLAY
0	1	0	1	A	
0	1	1	0	6	DISPLAYS MOVING LEFT TO RIGHT
0	1	1	1	E	
1	0	0	0	1	
1	0	0	1	9	RHS 7-SEGMENT DISPLAY

Fig. 7 (c)

Fig. 7: Organisation of display and keyboard. Display segments are driven by Port A of PIA 1, with cathodes selected by decoder outputs.



Component overlay of the DATUM board. PIA2 and ACIA are optional additions.

Assembling the DATUM kit

Start construction with the keypad. The kit provides a nut and bolt to fix the keypad to the board. When soldering the connecting pins of the keypad be wary of applying excessive heat, or else internal solder joints could melt and create an open circuit. Mount the six pushbutton switches then the resistors and capacitors.

Make sure that the electrolytic capacitors are correctly oriented as shown on the PCB overlay.

Mount the diode, LED and the seven-segment displays, again observing the polarity of each part. The 4MHz quartz crystal and the socket for the EPROM should be installed next. We also installed sockets for the microprocessor and the PIA1, although these are not provided in the kit.

Install the ICs, starting with the smaller packages, and making sure that pin 1 of each chip aligns with the 1 printed on the board. Before mounting the ICs, apply power to the circuit and check for the presence of 5V on the appropriate supply pins.

Install EPROM, PIA1 and microprocessor. Finally stick the rubber pads on the underside of the board to keep it clear of the bench.

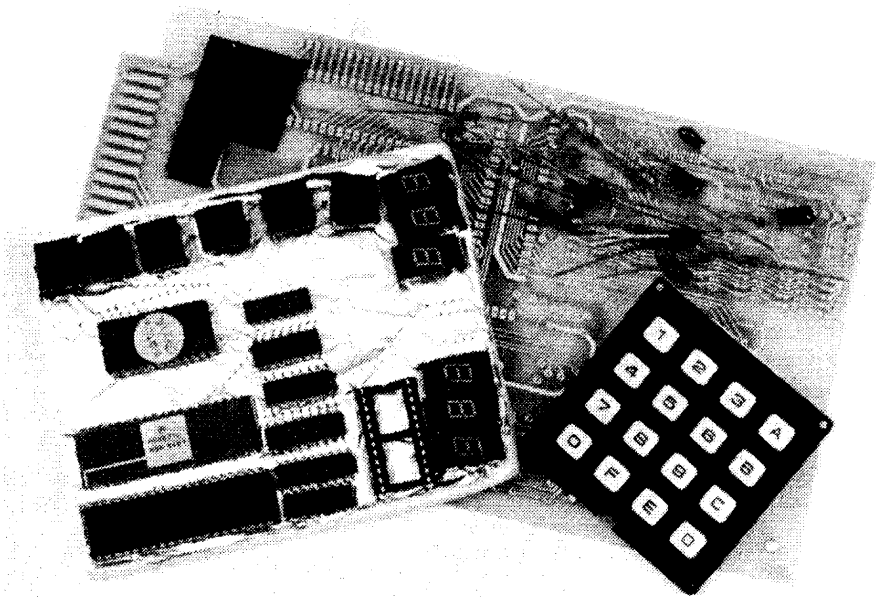
The manual that comes with the DATUM kit provides a fairly complete description of the circuitry from a technical point of view and step-by-step assembly instructions. A simple trouble-shooting guide is also provided which proved unnecessary in our case. The only problem we had was with the placement of capacitor C6, which is parallel to and directly opposite C7. An inviting pair of holes to the left are through-the-board connections, not for component installation.

We also found that it was necessary to cut the plastic supports off the bottom of the pushbutton switches supplied so that they would mount flush on the board. This makes it slightly difficult to mount the switches neatly and some care is required.

The PCB is not solder masked, and some of the tracks are closely spaced. A soldering iron with a small tip is essential here.

The manual provides an example of test procedures, including reading the contents of the EPROM and a simple program to test the RAM. If everything checks out OK you're on your way. Our next article will provide some software for the system and article number three will provide suggestions for applications.

Assembled or kit form DATUMs are available from Gammatron, Weens Road, Pooraka, 5095. Printed circuit boards, key pads and monitors are also available separately. Next month we shall look at software aspects, including the monitor routines.



DATUM kit provides PCB and all components for a minimal system. Full instructions are included.

UPPER (IC) SIDE OF PCB		PIN NUMBER COMMENCING FROM TOP	UNDER SIDE OF PCB
	+5V	1	0V AND COMMON
ACIA LINES (NOT USED)	RX DATA	2	
	RX CLOCK	3	
	TX CLOCK	4	
	$\overline{RTS}$	5	
	TX DATA	6	
	$\overline{DCD}$	7	
	$\overline{CTS}$	8	
	$\overline{IRQ}$	9	E
	$\overline{RESET}$	10	ADDRESS BLOCK 3000 (4K BYTES)
	A0	11	ADDRESS BLOCK 2000 (4K BYTES)
	A1	12	ADDRESS BLOCK 1000 (4K BYTES)
	A2	13	R/$\overline{W}$
	A3	14	VMA
	A4	15	D0
	A5	16	D1
	A6	17	D2
	A7	18	D3
	A8	19	D4
	A9	20	D5
	A10	21	D6
	A11	22	D7

Fib. 8: Connections for 44-pin expansion interface of DATUM. The third article in this series will cover applications.

DATUM: programming and monitor software

Last month we covered the design and construction of DATUM, a low-cost microcomputer designed to teach the basics of microprocessor systems. This article covers some of the software aspects, including the on-board Monitor routines and some sample programs.

First of all have a look at Fig. 1. This is a "programming model" of the MC6802 microprocessor, showing the internal registers of the chip which are available to the programmer. These registers are storage locations which are operated on by the various instructions of the microprocessor.

As shown in Fig. 1 the MC6802 has two 8-bit accumulators, A and B, shown at the top of the diagram The accumulators (abbreviated ACC) have instructions associated with them which can perform basic arithmetic and logical operations. All instructions that operate on an accumulator can be used on either ACC A of ACC B, except for one, Decimal Adjust Accumulator, used in converting binary to BCD. This instruction operates only on ACC A.

Next is the index register (IX). This register is 16-bits long and is usually used to store a 16-bit address which "points" to an item of interest, such as a memory location or an output port.

The Program Counter (PC) keeps track of the address of the current instruction, and is incremented automatically when the next instruction is required.

The Stack Pointer (SP) is also a 16-bit register, and stores the address of an area of memory defined by the user as a "stack". The stack is used by the processor as a temporary storage area to save the address to be returned to after executing a subroutine, or to save the contents of all registers (except the stack pointer) when an interrupt is encountered.

The stack pointer is set by the DATUM Monitor program to address 007F when the Reset key is pressed. The area of memory with addresses from 0000 to 007F is located on the MC6802 chip itself, and is used by the Monitor for temporary storage.

Besides its use in automatically saving subroutine return addresses, and register contents at an interrupt, the stack can be accessed by the programmer with "Push" and "Pull" instruction. These instructions must be used carefully to ensure that the stack is not disorganised. A common cause of "crashing" programs is incorrect use of the stack. If, for example, a data byte is left on top of the stack when the processor is expecting to find a subroutine return address, the program will be off into the never-never.

The final register shown in Fig. 1 is the Condition Code Register (CCR). This register indicates the state of the machine after it has executed an instruction. The state of each bit or a logical combination of bits in this register are used to determine the operation of conditional instructions. There are six separate conditions which will be indicated after every instruction;

H: Half Carry (a carry bit from bit 3 to bit 4 of a binary number)

I: Interrupt mask. Determines whether the processor will respond to a maskable interrupt

N: Indicates a negative number in two's complement binary

Z: Indicates a zero byte.

V: Overflow. Result is too large to be represented in 8 bit two's complement binary

C: Carry (a carry bit from bit 7 of a binary number)

Although a standard 8 bit register is used for condition codes, bits 6 and 7 are permanently set to "1" and can be ignored.

For more detailed discussion of 6802

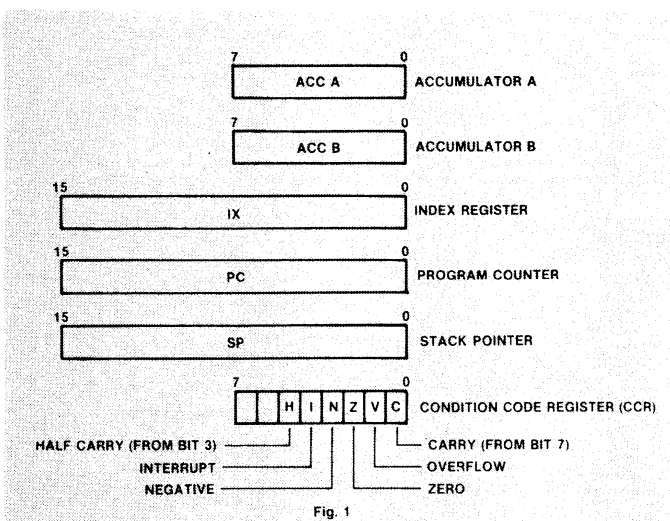


Fig. 1

Fig. 1 shows the programming model of the 6802 microprocessor. These are internal registers accessible by the user.

registers and programming, refer to our series of four articles on "How to Program in Machine Language", which began in EA, March 1982. The "M6800 Microprocessor Applications Manual", published by Motorola Inc is also a useful reference source.

What is a Monitor program?

A Monitor program is a collection of short programs that assist the user when communicating with the microprocessor system. The routines in the Monitor are directly related to the hardware of DATUM, and are called into action by the user's keypad input via a routine called the "command processor". This routine performs a number of tasks;

- Refreshing the display
- Scanning the keypad
- Checking the validity of the user's input
- Transferring to control to function routines when required.

Fig. 2 shows a simplifed version of the command processor program. After the machine has been reset it outputs a prompt "–" in the leftmost display digit. The keypad is scanned, the display refreshed and if no key has been pressed the MC6802 will continue to refresh the display and check the keypad.

When a key is struck a test is carried out to determine whether or not the key was valid. If the key was not valid the program returns to the display refresh routine without changing the display or the contents of any memory locations in the user area of the system. If however, the key was valid and, for example, it was a hexadecimal digit, the display would then show that digit. This is a simple example of multi-tasking, where a number of jobs are being carried out in sequence by the processor. To the user however, it looks as though this happens instantaneously.

If the key pressed was one of the command keys and it is valid then the MC6802 will start to execute a program that corresponds to that command.

An important function of the Monitor and probably the one most frequently overlooked is that of base conversion. The microprocessor has only one language, that of binary numbers. Humans have difficulty when dealing with such numbers so the hexadecimal number system (base 16) is used instead. There are at least two reasons why base 16 is most frequently used in computing:
(1) Conversion between base 2 and base 16 is mathematically a simple matter.
(2) People can easily remember numbers written in hexadecimal.

Another important feature of a Monitor are functions that allow the user to examine and change the contents of given memory locations so that programs can be stored. Equally important

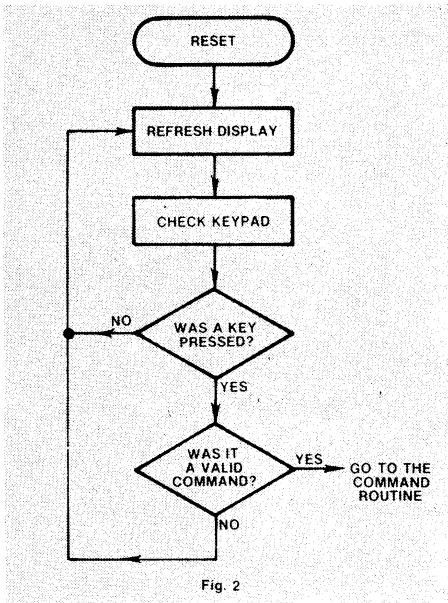


Fig. 2

Fig. 2 is a flowchart of the command processor program, responsible for refreshing the LED display, scanning the keypad and verifying the user's input.

Fig. 3 is a more detailed flowchart of the command routine, showing the actual names and addresses of the individual command routines in the DATUM Monitor ROM.

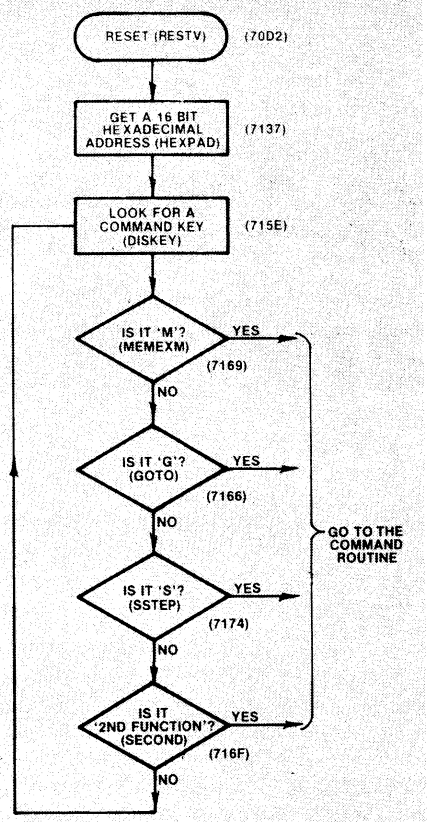


Fig. 3

is a "go" command, which allows the user's programs to be executed. Finally there should be some debugging aids that allow the programmer to trace through a program and so determine if the computation being performed is as intended.

Fig. 3 shows a flowchart similar to that of Fig. 2, with the difference that the actual names and addresses of the routines in the DATUM Monitor are also shown. Briefly, these routines operate as follows; shown with the name of "label", first, and the hexadecimal starting address shown in parentheses;

DISKEY (7155):	This subroutine refreshes the display and then scans the keypad.
GOTO (7166):	This routine causes a jump out of the Monitor to the user's program.
HEXPAD (7137):	This subroutine allows the user input a 4-digit hexadecimal number.
MEMEXM (7169):	This is the memory examine and change routine.
SECOND (716F):	This routine toggles the second function flag.
SSTEP (7174):	This is the single step or trace routine

Monitor commands of DATUM

This section describes the operation of each Monitor command in detail. Fig. 4 shows the layout of the keypad and display of DATUM and should be referred to as an aid to understanding how to "drive" the machine.

First of all we should consider the operation of the memory display and change function. This is one of the most important functions of the Monitor as without it programs cannot be loaded into the memory of DATUM. To display the contents of a given memory location all the user need do is to enter the address of the required memory location using the keypad and press the M/R key. The data digits (the two right-hand digits) of the display will then show the contents of that location. If the user wants to change the contents of the displayed location two new hexadecimal digits can now be entered from the keypad.

When entering new data into memory or just inspecting the current contents of a number of locations the I/D key is used to increment the displayed address. Some of the less obvious features of the memory function are that the user can input any number of hexadecimal digits, however only the two digits on the display just prior to the user pressing either the I/O or RESET or the ESCAPE key will be stored. If the Second Function

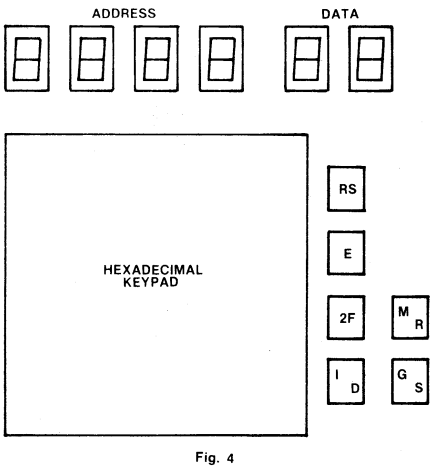
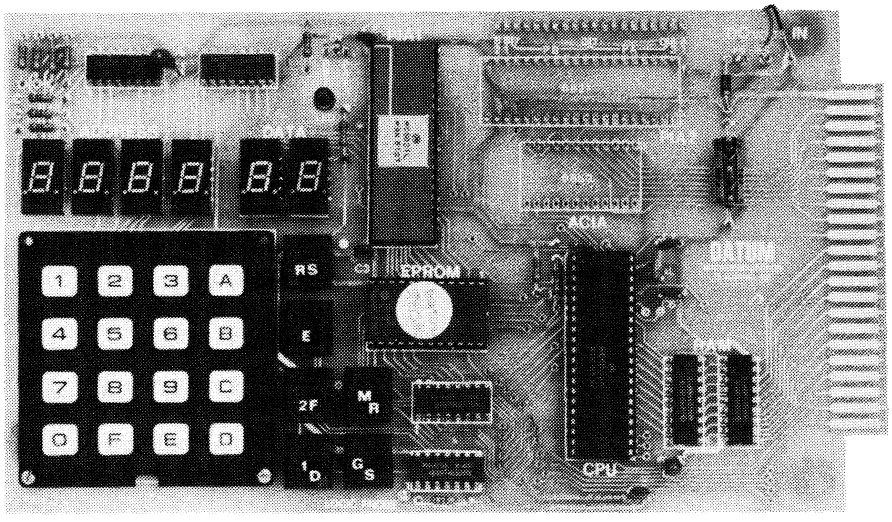


Fig. 4: layout of the DATUM function keys.



key is pressed once only before using the I/D key, the address will be decremented each time the I/D key is used. There are three ways of escaping from this decrement mode. The first is to use the 2nd function key once again, returning to the increment mode without exiting from the memory function. The other two ways are by using the RESET or the ESCAPE keys, which do cause an exit from the memory function.

When the contents of a memory location are changed the data display changes with every key stroke (every half byte, or "nibble"). If only one nibble is changed the contents of the memory location will then be the new high order nibble and the old low order nibble.

Once a program has been stored in memory the next Monitor command required is one that tells the machine to set the program counter to the start address of the user's program and to begin to execute it. The G/S key provides this function. Pressing this key after entering the four digit start address will cause the program to begin to execute from that address. However, the user may find that the first few attempts at writing software may not always produce results that are expected. The type of unpredictable results can vary from a value not being calculated correctly to a "crash" where all of the memory in which the program had been stored is over-written. If the unit does "crash" as just described then the prompt may not return when the ESCAPE key is pressed. In this case a master RESET is needed.

Obviously some debugging functions are also required. A trace or single step mode is one such tool. The single stepping mode is entered by keying in the start address of the program, pressing the 2nd function key and then the G/S key. The display will now show the start address and the opcode of the first instruction of the program. Now if the I/D key is pressed the address will be that of the next instruction and the data display will show the opcode of that instruction. In this mode none of the data bytes associated with these instructions are ever displayed. With this function programs can be checked one step at a time for correct operation.

When executing a program in this manner it is useful to inspect the contents of the registers, which can be done with the M/R key. In this case there is no need to press the 2nd function key before the M/R key because the 2nd function is already engaged in the single step mode.

For example, if the program shown in Fig. 5 is entered and run, examination of the registers will show the sum in Acc A, and the second addend in Acc B, where they were placed by the program. This illustrates an important point – the Store operation does not change the data in the source register, it merely copies it to the destination.

When the register display mode is entered the first register to be shown will be the condition code register. If the I/D key is then pressed a number of times the remainder of the registers will be displayed. Table 1 shows the order in which the registers are displayed, together with the actual display on DATUM. The two righthand displays are the register indentification, 'X' is the contents of a register and "–" is a blank display.

(CC)	XX––CC
(ACC B	XX––Ab
(ACC A)	XX––AA
(IX)	XXXX Id
(PC)	XXXX PC
(SP)	XXXX SP

Table 1: Order in which registers are displayed by the M/R function.

When the user has incremented the register display through to the stack pointer, pressing the I/D key once again will cause the condition codes to be displayed again. If the user wants to exit from the register display mode but continue single stepping, then the G/S key should be used. On returning to the single step mode the step just before entering the register display will be shown.

Now that the operation of the single step function has been described, we should next look at how it works so that the user can gain the full benefit from this function. As a piece of software it is the most complex section of the monitor. The function is performed by software only, there being no special hardware added to DATUM to perform the interrupt. It operates in the following manner:

(1) The opcode to be executed is checked to determine the number of byts in the instruction. If it is a branch or a jump instruction the destination address is determined.

(2) The single step program now knows where the target program will go to next, so the instruction in that location is stored away in the scratch pad memory and a software interrupt (SWI) is entered in its place. The opcode of this instruction is "3F".

(3) The instruction is executed and then the processor will reach the software interrupt which will cause the processor to resume execution of the single step program.

(4) Finally the software interrupt is replaced by theinstruction opcode that was stored away. The above process is then repeated for the next instruction.

It should be noted that this type of single step will only work in a read/write memory. In fact if the operator tries to use this command to trace through the DATUM monitor EPROM the prompt will return to the display.

Some attention must be given to the subject of breakpoints. A breakpoint is another debugging tool, an instruction that is placed into the user's program to terminate the program when it is executed. By using this function programs that have long execution times such as those using delay loops can be executed at full speed till the breakpoint is reached. Due to the small number of function keys on DATUM is was decided that a breakpoint function could not be incorporated. However, the user can still have this if the following steps are taken.

Determine where in the program a breakpoint is needed (this at first may be only a guess) and change the opcode of of the selected instruction to a "3F" for a software interrupt (SWI) (note down the original opcode for future reference). Run the program in the normal way and if it is correct and there are no very long delays in the program the displays should immediately light up with the address of the SWI and "3F" in the data display.

If the display does not return in reasonable time it can be assumed that the program has crashed and that the SWI should be placed closer to the start of the algorithm.

When SWI is being displayed the user can press the 2nd function and then the M/R key to display the register contents at this point of the program.

Having finished with this particular breakpoint the operator must replace the SWI opcode with the original code used in the program.

When attempting to use the register display function it should be noted that DATUM will only display registers after a SWI has been executed by the processor. This has been done deliberately so that only valid register contents are displayed.

Finally there is a base conversion package which is located in the monitor EPROM. There are three routines which allow the user to convert from hexadecimal to binary, octal, or decimal. In each case when these routines are running the prompt is moved to the second display from the right and every time a new hexadecimal value is keyed in on the right-hand side the result is displayed on the four left-hand side digits. To run these routines, key in relevant address and press the G/S key. The program address are hex 7500 for hexadecimal to binary, 7503 for hexadecimal to decimal, and 7506 for hexadecimal to octal.

```
1010 B6 10 00  START   LDA A  SUM1      LOAD ACC A WITH THE FIRST VALUE
1013 F6 10 01          LDA B  SUM2      LOAD ACC B WITH THE SECOND
1016 1B                ABA              ADD THE TWO ACCS. TOGETHER
1017 B7 10 02          STA A  SUM3      SAVE THE RESULT
101A 3F                SWI              STOP
```

Fig. 5: A short program to add two numbers together. The first four digits are address locations at which the corresponding op codes and data are entered.

```
1000 BD 71 07  START   JSR    PROMPT    CLEAR DISPLAY AND OUTPUT A PROMPT
1003 CE 00 10  LOOP1   LDX    #$10      SET DELAY TIME
1006 BD 71 DD  LOOP2   JSR    DISPLY    GO TO THE DISPLAY ROUTINE
1009 09                DEX              DECREMENT THE X REG
100A 26 FA             BNE    LOOP2     IS X REG ZERO?
100C 96 10             LDA A  DISBUF    LOAD ACC A WITH DISBUF CONTENTS
100E 81 FF             CMP A  #$FF      IS IT A BLANK?
1010 26 03             BNE    INCDIS    IF NOT BLANK GO TO INCDIS
1012 4A                DEC A            BLANK, DECREMENT FOR A PROMPT
1013 20 01             BRA    STORIT    SKIP INCREMENT
1015 4C        INCDIS  INC A            PROMPT, INCREMENT FOR A BLANK
1016 97 10     STORIT  STA A  DISBUF    STORE NEW DISPLAY VALUE
1018 20 E9             BRA    LOOP1     BRANCH TO DISPLAY/DELAY LOOP
                *
                *
                **     IF DISBUF = FE THEN A PROMPT IS DISPLAYED
                **     IF DISBUF = FF THEN THE DISPLAY IS BLANK
                *
                *
                       END
```

Fig. 6: A program to flash the prompt on the LED display, showing the use of the Monitor routines SPROMPT and DISPLAY.

Programs for DATUM

This subject will be covered in more detail in the next article, however, by way of introduction a few programs will be presented.

The first is a very simple program to add two hexadecimal numbers together and has been included to demonstrate the operation of the single step mode. The program listing in Fig. 5 should be placed in memory starting at location 1010 (hex) with the two numbers to be added placed in locations 1000 and 1001. Values that will be used for this example are 10 and 20 respectively. Once these values and the program have been placed in memory the user should begin the single step mode at location 1010. At this point the display will show the address 1010 and the contents of the location (data) equal to B6. If the I/D key is pressed four times the opcodes of the instructions and the corresponding addresses will be displayed as in Table 2.

Now that the program has been executed the user can check location 1002 to see if the correct value, 30, has been stored. If the user wants to display the registers while in the single step mode then the method described above should be used.

HEX ADDRESS	DATA
1010	B6
1013	F6
1016	1B
1017	B7
101A	3F

Table 2: Addresses and data for the addition program shown in Fig. 5.

The next program, listed in Fig. 6, flashes the prompt segment on the display. This program uses two subroutines that reside in the Monitor. The first, "SPROMPT" is located at 7107 and it has the function of clearing the display then placing the prompt character in the display buffer. The second is called "DISPLAY" and is the display multiplex subroutine.

The flash rate may be varied by altering the value loaded into the index register at the beginning of LOOP1, that is, the 16-bit number in memory locations 1004 and 1005. Currently it is 0010, but if it is reduced the flash rate will increase. The final listing, Fig. 7, is that of a 12 hour clock. When running, this program prompts the user with a lower case "t" and waits till the time has been entered. If a non-decimal number is entered or an invalid digit is keyed in, then the incorrect digit will be set to zero. The program requires inputs of hours and minutes only, as the seconds are automatically set to zero. If the hours figure is less that 9 then a leading zero must be entered in the tens of hours digit.

The clock program is shown overleaf. The first four digits are addresses, followed by the op codes and data to be entered.

In the next article more programs and applications will be presented, including games and control applications for the hobbyist.

DATUM microcomputer: 12 hour clock program

```
                        *
                        **        TIME SETTING ROUTINE
                        *
                        *
1000 860E     CLOCK     LDAA    #$0E          PROMPT WITH A "t"
1002 9710               STAA    DISBUF
1004 BD7137             JSR     HEXPAD        WAIT FOR TIME INPUT
1007 CE0016             LDX     #INPUTST      POINT INPUT STORE
100A 6F19               CLR     XOFFS+4,X     CLEAR TENS AND UNITS
100C 6F1A               CLR     XOFFS+5,X     OF SECONDS
100E A600     COPYTM    LDAA    0,X           COPY INPUT TO CURRENT
1010 A715               STAA    XOFFS,X       TIME STORAGE
1012 08                 INX
1013 8C001A             CPX     #INPUTST+4
1016 26F6               BNE     COPYTM
1018 8D26               BSR     MISCLR
101A 8109               CMPA    #9            CHECK INPUT FOR A VALID
101C 8D1E               BSR     INTEST        TIME, IF MINUTES ARE NOT
101E 8105               CMPA    #5            VALID SET TO ZERO, IF HOURS
1020 8D1A               BSR     INTEST        ARE NOT VALID SET TO
1022 8109               CMPA    #9            1 O'CLOCK
1024 8D16               BSR     INTEST
1026 274B               BEQ     DISTM
1028 8101               CMPA    #1
102A 2608               BNE     SET100
102C A601               LDAA    1,X
102E 8102               CMPA    #2
1030 2202               BHI     SET100
1032 203F               BRA     DISTM
1034 6F15     SET100    CLR     XOFFS,X
1036 6F16               CLR     XOFFS+1,X
1038 6C16               INC     XOFFS+1,X
103A 2037               BRA     DISTM
                        *
                        *

                        *
103C 2302     INTEST    BLS     MISCLR
103E 6F15               CLR     XOFFS,X
1040 09       MISCLR    DEX
1041 A615               LDAA    XOFFS,X
1043 39                 RTS
                        *
                        *

                        **        MAIN TIME DISPLAY ROUTINE
                        *
                        *
1044 8633     CLCDIS0   LDAA    #51           SET DELAY LOOP, ACCA
1046 CE001B             LDX     #27           "    "    "  , X REG
1049 BD71D7             JSR     XTIMLP        JUMP TO X REG DELAY
104C 36       CLCDIS1   PSHA                  SAVE ACCA
104D BD71DD             JSR     DISPLAY       REFRESH THE DISPLAY
1050 32                 PULA                  GET ACCA DELAY VALUE
1051 4A                 DECA                  DECREMET ACCA
1052 26F8               BNE     CLCDIS1       IS ACCA ZERO?
```

```
1054 CE0030             LDX     #TEMPMEM+5    POINT TO UNITS OF SECONDS
1057 8D38               BSR     BUMPTM
1059 8D33               BSR     BUMPT9        CHECK UNITS OF SECONDS
105B 8D35               BSR     BUMPT5        CHECK TENS OF SECONDS
105D 8D2F               BSR     BUMPT9        CHECK UNITS OF MINUTES
105F 8D31               BSR     BUMPT5        CHECK TENS OF MINUTES
1061 8103               CMPA    #3            CHECK FOR 12 O'CLOCK
1063 2704               BEQ     TENTST
1065 8D27     TENRET    BSR     BUMPT9        CHECK UNITS OF HOURS
1067 200A               BRA     DISTM
1069 E600     TENTST    LDAB    0,X
106B 27F8               BEQ     TENRET
106D 6F00               CLR     0,X
106F 8601               LDAA    #1
1071 A701               STAA    1,X

                        *
                        **        TIME DISPLAY ROUTINE
                        *
                        *
1073 CE0010   DISTM     LDX     #DISBUF           POINT TO DISBUF
1076 A61B     DISTM1    LDAA    TEMPMEM-DISBUF,X  READ CURRENT TIME
1078 BD727D             JSR     BIN7SEG           GET SEVEN SEG DATA
107B A700               STAA    0,X               STORE IT IN DISBUF
107D 08                 INX                       INC POINTER
107E 8C0016             CPX     #DISBUF+6         IS IT END OF BUFFER?
1081 26F3               BNE     DISTM1
1083 7D0002             TST     TEMPMEM           IS FIRST LOCATION ZERO?
1086 2604               BNE     DISTM2
1088 86FF               LDAA    #$FF              REMOVE LEADING ZERO
108A 9710               STAA    DISBUF
108C 20B6     DISTM2    BRA     CLCDIS0           RETURN

                        *
                        *
                        **        CHECK AND INCREMENT TIME
                        *
                        *
108E 8109     BUMPT9    CMPA    #9
1090 2002               BRA     BUMPTM
1092 8105     BUMPT5    CMPA    #5
1094 2309     BUMPTM    BLS     NOINC
1096 6F01               CLR     1,X               CLEAR "UNITS"
1098 A600     BUMPTS    LDAA    0,X               GET "TENS"
109A 4C                 INCA                      INCREMENT "TENS"
109B A700               STAA    0,X
109D 09                 DEX                       DECREMENT POINTER
109E 39                 RTS                       RETURN
109F 31       NOINC     INS                       RESTORE STACK POINTER
10A0 31                 INS
10A1 20D0               BRA     DISTM             GO TO DISTM
```

DATUM: games and software design

DATUM, a "Digital Aid for Teaching you Microprocessors", is a single board microprocessor trainer based on the MC6802 processor. Construction and programming have been covered in previous articles. This final instalment provides more programming details, including three useful examples to show what can be done.

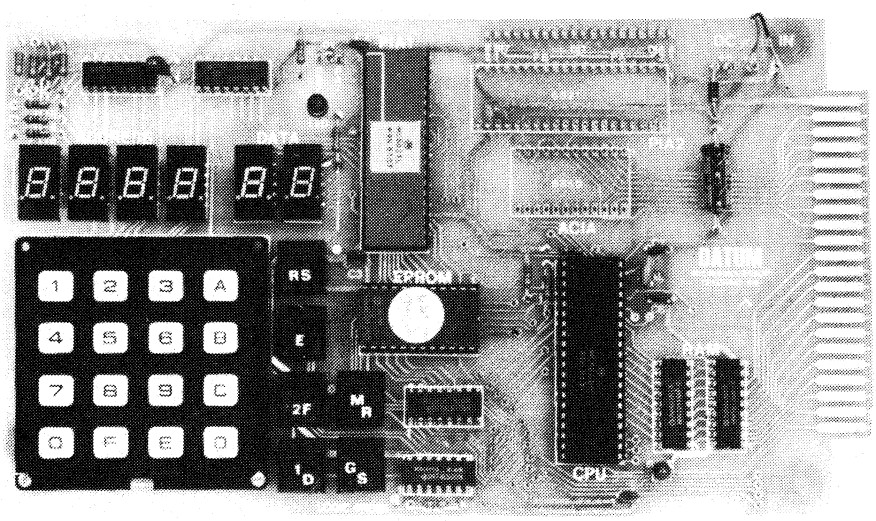
Although called a minimal microprocessor system, DATUM is actually quite powerful, with applications limited only by the user's imagination and programming abilities. Skill in programming comes with experience, but to get things started this article has a few examples. They have been written as games, but lend themselves readily to more practical applications.

Before examining the programs however, we should point out that many useful routines have already been written and incorporated in the DATUM monitor. Time delays, character display and keyboard scanning routines are available to be incorporated as subroutines in your own programs.

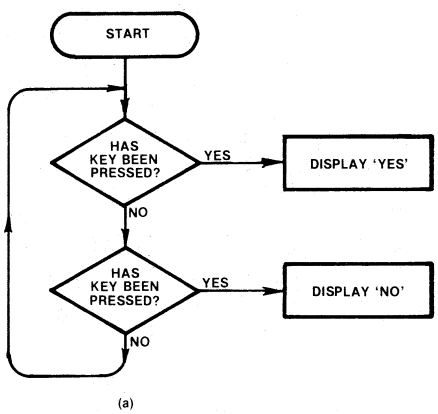
Table 1 provides the names and starting addresses of useful subroutines in the DATUM monitor and comments on their use. There is a penalty for using them, however – single stepping through ROM routines is not possible. A single-step jump to a monitor routine simply brings up a prompt and halts execution. In some cases this can be overcome by copying the monitor routine into RAM, with appropriate address changes.

An alternative method of debugging programs which incorporate monitor routines is to single step up to the point of the jump to the monitor subroutine, reset and then recommence single stepping at the instruction following the subroutine call. We can, after all, assume that the monitor subroutine itself is correct.

When writing programs it is good practice to finish with a software interrupt instruction (3F). Should there be an error in



Repeated from last issue, this photo shows the completed DATUM microprocessor board. Construction and Monitor software have been covered in previous articles.



(a)

Fig. 1 (a) shows the flowchart for a simple decision-maker program loop.

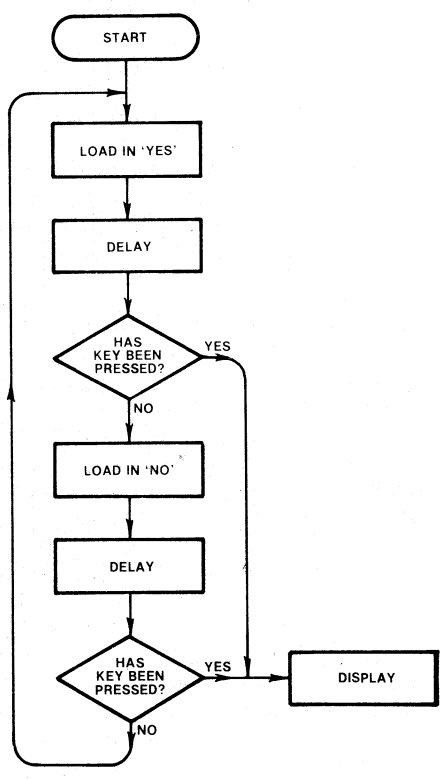


Fig. 1 (b)

Fig. 1(b) shows a more complex approach to the same problem which is more flexible, allowing the use of common subroutines.

the program then this instruction may stop the program running into higher memory locations and overwriting your program. With these remarks out of the way we can discuss the example programs.

A decision maker

This simple decision maker program consists of a loop that is interrupted by depressing any of the hex keys. The flowchart of Fig. 1(a) shows that the program consists of two decisions, represented by the diamond boxes. Depending on the exact time a key is depressed, the answer is either "yes" or "no". Fig. 1(b) shows a more complex approach which has a number of advantages. By loading in the word to be displayed before a decision has to be made, a common display subroutine can be used. The program could be organised so that the delay and decision blocks can be shared, but since these are subroutines within the monitor that we will call upon there is little point in doing this.

By introducing delay blocks we can weight the "yes-no" decisions depending on the relative length of each delay. In this program we will make them equal to give a 50:50 chance for the answers to be "yes" or "no". Finally, both flow diagrams can readily be extended to become higher order decision makers, with other words like "stop", "danger" being displayed.

The next question is how the flow chart of Fig. 1(b) is converted into an actual program. Each box represents a small program, or "module" which must be written. We will consider each function in turn.

Firstly we have to load the display register with hex numbers that provide the appropriate characters for our message when they are decoded by the display drivers. From the monitor listing (supplied with DATUM kits) it can be seen that a range of characters is available in a "Display look-up table" at lines 77A0 to 77C1.

Also from the monitor listing we can see that memory locations 0010 to 0015 are the six display register locations. Whatever is loaded into memory location 0010 is displayed on the first seven segment display, with the contents of memory location 0011 displayed as the second digit and so on (all addresses are in hexadecimal).

Loading of the word "no" is performed in the same way except that only the first three digits have to be changed, since the last three are already blanked.

To display the word "yes" we must load location 0010 with 48, the hex code for "y", location 0011 with 06 ("E"), and location 0012 with 42 ("S"). Locations 0013 to 0015 are loaded with hex 7F, which is the code for a blank. The first part of the decision maker program in listing 1 uses the index register to address the display registers one by one.

A short time delay is required by the second module of the program, and this can be most easily achieved by using the timing subroutines in the monitor. A jump to TIMLP at address 71D2 will provide a 10ms delay. Larger delays can be generated by using additional loops to call TIMLP as many times as required. Alternatively by using another timing subroutine XTIMLP at memory location 71D7, we can provide a delay equal to 8 multiplied by the value in the X register, in microseconds.

Testing whether or not a key has been depressed can be done by using the monitor subroutine MPXK at address 7177. At the end of this subroutine a non-zero number is loaded into accumulator B if a key has been pressed while the value of the key is in accumulator A.

In this case we are not interested in the particular key, but simply whether or not a key has been depressed. Thus, if accumulator B has zero contents, no key has been depressed and we must continue on in the loop. However, if the contents of accumulator B are non-zero, a key has been depressed and the appropriate "decision" must be displayed. Listing 1 thus shows a jump to test if a key has been depressed (to location 101A) and the subsequent statements to test whether the contents of accumulator B are zero or not.

Program listing 1: Decision maker

```
0010                        DISBUF  EQU     $10     DISPLAY BUFFER MEMORY
71DD                        DISPLY  EQU     $71DD   DISPLAY REFRESH
7177                        MPXK    EQU     $7177   KEYBOARD SCAN
71D2                        TIMLP   EQU     $71D2   10MS TIME DELAY
                            *
1000                        *       ORG     $1000
                            *
                            **      SET UP TO DISPLAY 'YES'
                            *
1000 CE 00 10               YESSET  LDX             #DISBUF POINT TO DISBUF
1003 86 48                          LDA     A       #$48    LOAD ACC A WITH A 'Y'
1005 A7 00                          STA     A       0,X     PUT IT IN THE 1ST DIGIT
1007 86 06                          LDA     A       #6      LOAD ACC A WITH AN 'E'
1009 A7 01                          STA     A       1,X     PUT IT IN THE 2ND DIGIT
100B 86 42                          LDA     A       #$42    LOAD ACC A WITH AN 'S'
100D A7 02                          STA     A       2,X     PUT IT IN THE 3RD DIGIT
100F 86 7F                          LDA     A       #$7F    LOAD ACC A WITH A
                                                            BLANK
1011 A7 03                          STA     A       3,X
1013 A7 04                          STA     A       4,X
1015 A7 05                          STA     A       5,X
1017 BD 71 D2                       JSR             TIMLP   WAIT FOR 10 MS
                            *
                            **      HAS A KEY BEEN PRESSED?
                            *
101A BD 71 77                       JSR             MPXK    SCAN KEYBOARD
101D 26 19                          BNE             OUTPUT
                            *
                            **      SET UP DISPLAY FOR 'NO'
                            *
101F CE 00 10                       LDX             #DISBUF POINT TO DISBUF
1022 86 3A                          LDA     A       #$3A    LOAD ACC A WITH AN 'N'
1024 A7 00                          STA     A       0,X     PUT IT IN THE 1ST DIGIT
1026 86 01                          LDA     A       #1      LOAD ACC A WITH AN 'O'
1028 A7 01                          STA     A       1,X     PUT IT IN THE 2ND DIGIT
102A 86 7F                          LDA     A       #$7F    LOAD ACC A WITH A
                                                            BLANK
102C A7 02                          STA     A       2,X
102E BD 71 D2                       JSR             TIMLP   WAIT FOR 10 MS
                            *
                            **      HAS A KEY BEEN PRESSED?
                            *
1031 BD 71 77                       JSR             MPXK    SCAN KEYBOARD
1034 26 02                          BNE             OUTPUT
1036 20 C8                          BRA             YESSET  SET UP FOR 'YES' AGAIN
                            *
                            **      OUTPUT THE DECISION AND FREEZE THE DISPLAY
                            *
1038 86 80                  OUTPUT  LDA     A       #$80    SET DISPLAY FREEZE TIME
103A B7 10 48                       STA     A       TEMP    SAVE IT
103D BD 71 DD               OUT1    JSR             DISPLY  REFRESH THE DISPLAY
1040 7A 10 48                       DEC             TEMP
1043 26 F8                          BNE             OUT1    IS IT FINISHED?
1045 20 B9                          BRA             YESSET  STAR AGAIN
1047 3F                             SWI
                            *
1048                        TEMP    RMB             1       FREEZE TIME
                            *
                                    END
```

The first five lines of this listing are information only. The program code starts at address 1000 and each line contains two or three bytes of code which are loaded into successive addresses. The program ends at address 1047, with location 1048 used as temporary storage.

DATUM: Flowchart for Program 2

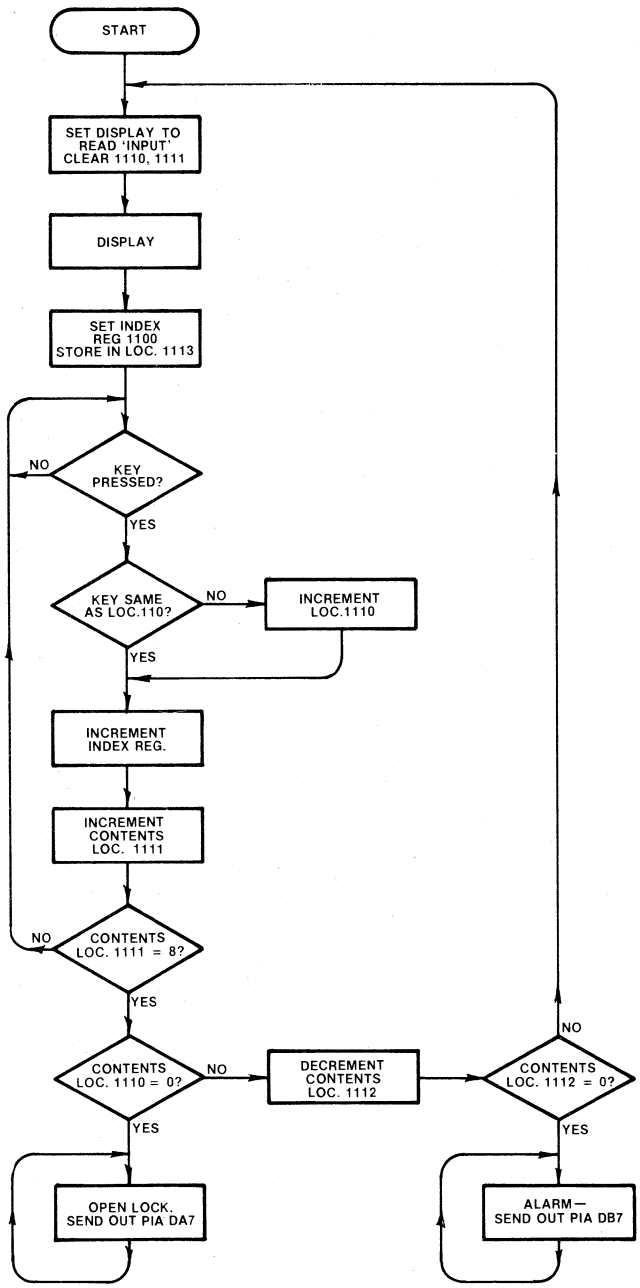


Fig. 2

The final box in the flow diagram is for displaying the characters already set up in the display buffers. Again a monitor subroutine can be employed, namely DISPLAY (71DD). This particular routine only displays a character for a few milliseconds so a small loop is introduced to hold the display for a longer period. Since DISPLAY makes use of both the A and B accumulators, the display counter is stored in memory location 1060. In the program given in listing 1 the value fed in is 80, providing a display for about 2½ seconds. Although not previously mentioned, a further advantage of flow diagram (1b) is that when the display is terminated the decision maker is immediately ready for another decision.

Combination lock

The second program may be considered as a guessing game but it also provides the basis for a combination lock system. An N-bit code are stored in the memory and when the user keys in the correct code the lock is energised and allowed to be open. In this example N is set equal to 8 but the value can readily be changed. If someone tries to break the code they are allowed three attempts before an alarm operates. Outputs to the lock and alarm are through the PIA data lines DA7 and DB7 respectively at the top of the board. Values on these lines can be confirmed using the logic probe. Fig. 2 shows the flow diagram for the program.

The majority of the programming routines have been discussed previously, with the exception of outputting a signal to the PIA. The eight digits of the correct combination are stored in memory locations 1100 to 1107 by the operator before the program is run. The number of errors in entering the combination is stored in location 1110, the number of times the keys are pressed per try in 1111 and the number of attempts to input the correct code is stored in location 1112.

Because the key input subroutine makes use of the index register, the contents of this register must be saved prior to calling the input subroutine. Memory locations 1113 and 1114 are used for this purpose. Listing 2 shows the complete program.

The PIA may be divided into two nearly identical halves, A and B. Each has three registers, the control register, the data direction register ("0" = input a signal; "1" = output a signal) and the actual data register. Only two address lines are used for each half of the PIA, the data direction and data registers sharing one address. The addresses used in DATUM for the display PIA are given in Table 2. To decide between the two registers that have a common address, bit 2 of the control register is employed. A zero in bit 2 allows the data direction register to be addressed while a 1 addresses the data register. Thus the sequence in setting up the A half of the PIA for outputting a signal is as follows:

- Set up the control register (address 6001) with a 0 in bit 2 position.
- Next set the data direction register (address 6000) to all 1's for output of data.
- Readdress the control register now putting a 1 in bit 2 position so that the data register will now be address at 6000.
- Finally, send the data out.

This sequence is used twice in the program, once to open the lock if the correct code is fed in and the second time to initiate the alarm signal. Notice that after sending information to the PIA the program jumps back to hold this instruction. If this is not done the monitor returns the display PIA to its normal role and the output immediately goes high again.

Should you wish to develop this program further for use as a safe lock or something similar, then the hex key pad

can be removed from DATUM and mounted remotely. The system should be organised so that power must be applied to undo the lock while removing the power will sound the alarm. Thus the system is secure should power to DATUM fail.

Pick up sticks

The final program is a game you may have played when younger. There is a pile of match sticks and the two players are allowed to remove in turn a number of matches (for example any number between 1 and 10), each player trying to force the other to pick up the last match. This simple game can be expressed in mathematical form and so this program, among other things, is to illustrate how DATUM can be programmed to perform simple arithmetic.

The easiest way to understand the winning strategy is to start from the end of the game and work backwards. If the maximum number of sticks that can be picked up by either player is M, then we wish to force things so that on our last move we leave a single match. Thus on our next to last move we leave (M + 1) + 1 matches, and on our third to last move 2(M + 1) + 1.

Starting with N matches in the pile for our turn, we must therefore leave

$$L = A(M + 1) + 1$$

matches, where A is the largest positive integer and is given by

$$A = \text{integer } [(N-1)/(M+1)]$$

Thus for any move the player should remove

$$R = N - A(M + 1) - 1$$

A problem occurs if the operator knows this strategy and also applies it. When this happens R equals zero and a check for this must be made. In this case DATUM subtracts one match stick in the hope that it was by chance that this situation arose and on the next time around he will win.

To set up the game on DATUM the operator feeds in N, a two-digit number, being the number of matches in the pile and M, and a single digit number, the maximum number of matches a player can pick up per turn. The operator is given the privilege of having first go and so they feed in P, the number of matches they wish to remove. The program checks the value to see they are not cheating. In fact, "DATUM" takes a rather sadistic attitude in this program. If he wins he calls the operator a CLOT and if he loses he says the operator has cheated.

Fig. 3 presents one flow diagram for the game Pick Up Sticks and listing 3, shows the program. The only difficult programming steps in the flow diagram are the compute stages. Division is achieved by multiple subtraction and multiplication by successive additions. The program as written has a number of defects and it is suggested that as an exercise you may try and make some changes. Firstly, the arithmetic is all done in hexadecimal and while this will provide practice for working out branch offsets, it would be more convenient to do this with decimal figures.

In addition to testing the input to restrict values from 0 to 9, decimal arithmetic requires the use of the Decimal Adjust instruction to allow for overflow. This instruction only works in conjunction with the three addition instructions ABA, ADD and ADC, so for decimal subtraction the values must be converted to two's complement and then added.

The program as written does make use of FOUR additional monitor subroutines

Text continues on P95

Flowchart for Program 3

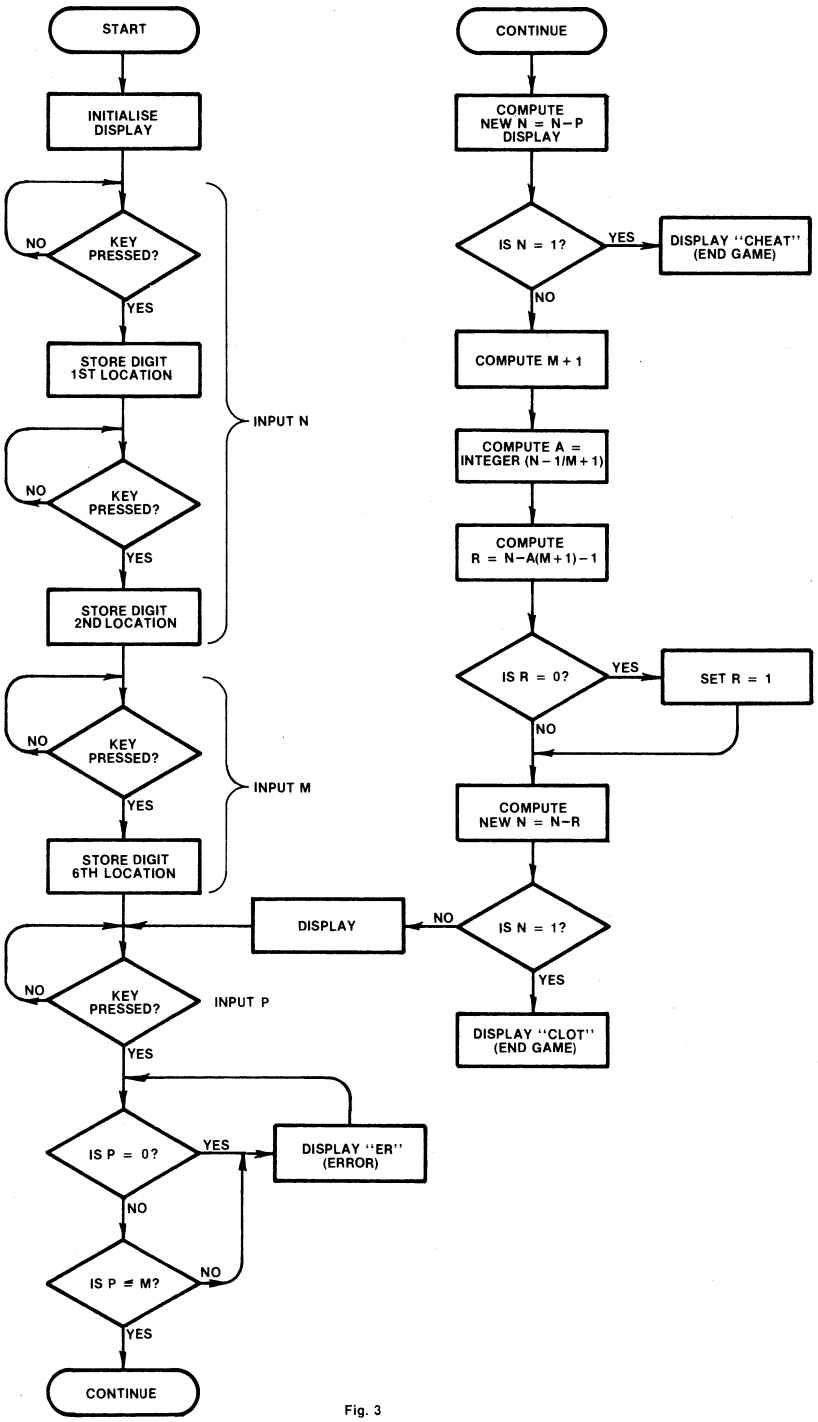


Fig. 3

DATUM: listing 2 — Combination lock

```
7107                    BLANK   EQU     $7107   CLEAR THE DISPLAY
0010                    DISBUF  EQU     $10     DISPLAY BUFFER MEMORY
71DD                    DISPLAY EQU     $71DD   REFRESH THE DISPLAY
7177                    MPXK    EQU     $7177   KEYBOARD SCAN
                        *
                        **      PIA ADDRESSES
                        *
6000                    PIAAD   EQU     $6000   A SIDE DATA/DDR
6001                    PIAAC   EQU     $6001   A SIDE CONTROL
6002                    PIABD   EQU     $6002   B SIDE DATA/DDR
6003                    PIABC   EQU     $6003   B SIDE CONTROL
                        *
1000                            ORG     $1000
                        *
1000 86 03              START   LDA A   3       SET THE NUMBER OF
1002 B7 11 0A                   STA A   NUMTRY  TRIES TO 3
                        *
                        **      DISPLAY THE WORD INPUT
                        *
1005 BD 71 07           INPUT   JSR     BLANK
1008 CE 00 10                   LDX     #DISBUF POINT TO DISBUF
100B 86 1F                      LDA A   #$1F    'I'
100D A7 00                      STA A   0,X
100F 86 91                      LDA A   #$91    'N'
1011 A7 01                      STA A   1,X
1013 86 14                      LDA A   #$14    'P'
1015 A7 02                      STA A   2,X
1017 86 09                      LDA A   #9      'U'
1019 A7 03                      STA A   3,X
101B 86 0E                      LDA A   #$E     'T'
101D A7 04                      STA A   4,X
101F 7F 11 08                   CLR     ERRCNT  CLEAR ERROR COUNT
1022 7F 11 09                   CLR     KEYCNT  CLEAR KEY COUNT
                        *
                        **      DISPLAY THE WORD 'INPUT'
                        *
1025 86 80                      LDA A   #$80    SET DISPLAY TIME
1027 B7 11 0B                   STA A   XTEMP   SAVE IT
102A BD 71 DD           DISOUT  JSR     DISPLAY REFRESH THE DISPLAY
102D 7A 11 0B                   DEC     XTEMP   DECREMENT DISPLAY TIME
1030 26 F8                      BNE     DISOUT  IS IT FINISHED?
                        *
                        **      GET THE COMBINATION
                        *
1032 CE 11 00                   LDX     #COMBIN POINT TO COMBINATION
1035 FF 11 0B                   STX     XTEMPT  SAVE THE POINTER
1038 BD 71 77           GETKEY  JSR     MPXK    SCAN KEYBOARD
103B 27 FB                      BEQ     GETKEY  HAS A KEY BEEN PRESSED?
103D FE 11 0B                   LDX     XTEMPT  POINT TO COMB. STORE
1040 A1 00                      CMP A   0,X
1042 27 03                      BEQ     MISERR  WAS IT AN ERROR?
1044 7C 11 08                   INC     ERRCNT  YES
1047 08                 MISERR  INX             NO, GO TO
                                                NEXT DIGIT
1048 FF 11 0B                   STX     XTEMP   SAVE POINTER
104B 7C 11 09                   INC     KEYCNT
                        *
                        **      CHECK KEY COUNT
                        *
104E C6 08                      LDA B   #8
1050 F1 11 09                   CMP B   KEYCNT  WAS THAT THE LAST KEY?
1053 26 E3                      BNE     GETKEY
1055 7D 11 08                   TST     ERRCNT  IS ERROR COUNT ZERO (MAX)
1058 26 12                      BNE     TRIES
                        *
                        **      COMBINATION IS OK, OPEN LOCK
                        *
105A 4F                         CLR A           SET PIAAD TO DDR
105B B7 60 01                   STA A   PIAAC
105E 43                         COM A           SET DDR FOR ALL OUTPUTS
105F B7 60 00                   STA A   PIAAD
1062 B7 60 01                   STA A   PIAAC   RESET PIAAD TO DATA
1065 86 47                      LDA A   #$47
1067 B7 60 00                   STA A   PIAAD   SET PA7 TO '0'
106A 20 FE              OPEN    BRA     OPEN    STAY OPEN
                        *
                        **      TEST NUMBER OF TRIES
                        *
106C 7A 11 0A           TRIES   DEC     NUMTRY
106F 7D 11 0A                   TST     NUMTRY
1072 26 91                      BNE     INPUT   IF NOT ZERO TRY AGAIN
                        *
                        **      SET ALARM
                        *
1074 4F                         CLR A           SET PIABD TO DDR
1075 B7 60 03                   STA A   PIABC
1078 43                         COM A           SET DDR FOR ALL OUTPUTS
1079 B7 60 02                   STA A   PIABD
107C B7 60 03                   STA A   PIABC   RESET PIABD TO DATA
107F 86 60                      LDA A   #$60
1081 B7 60 02                   STA A   PIABD   SET PB7 TO '0'
1084 20 FE              ALARM   BRA     ALARM
                        *
                        **      TEMPORARY STORAGE
                        *
1100                            ORG     $1100
                        *
1100                    COMBIN  RMB     8       THE COMBINATION SHOULD
                        *                       BE STORED HERE
1108                    ERRCNT  RMB     1       ERROR COUNT
1109                    KEYCNT  RMB     1       KEY COUNT
110A                    NUMTRY  RMB     1       NUMBER OF TRIES
110B                    XTEMP   RMB     2       TEMP STORE
                        *
```

This program can be used as a guessing game or as the basis for a security lock system. An 8-bit code is stored in memory before the program is run. The user has three chances to enter the correct code before an alarm connected to PIA1 bit DB7 is activated. All programs shown are entered in memory beginning at location 1000 (hex). Code before this address are "equates" which define the location of routines and storage used by the program, and should not be entered.

DATUM: listing 3 — Pick up sticks

```
722D                        BINSEG  EQU         $727D   'BIN7SEG' ON DATUM
7107                        BLANK   EQU         $7107   CLEAR THE DISPLAY
724D                        BYTE    EQU         $724D   FORM A BYTE FROM 2 NIB-
                                                        BLES
715E                        DISKEY  EQU         $715E   REFRESH DISPLAY AND
                            *                           SCAN THE KEYBOARD
0010                        DISBUF  EQU         $10     DISPLAY BUFFER
71DD                        DISPLAY EQU         $71DD   REFRESH THE DISPLAY
726E                        TWODIG  EQU         $726E
                            *
                            **      SET UP THE DISPLAY
                            *
1000                                ORG         $1000
                            *
1000 BD 71 07               START   JSR         BLANK   CLEAR THE DISPLAY
1003 CE 00 10                       LDX         #DISBUF POINT TO DISBUF
1006 86 7E                          LDA     A   #$7E    '~'
1008 A7 01                          STA     A   1,X
100A A7 05                          STA     A   5,X
                            *
                            **      GET AN INPUT
                            *
100C BD 71 5E                       JSR         DISKEY
100F B7 12 00                       STA     A   TEMP1
1012 BD 72 7D                       JSR         BINSEG
1015 97 10                          STA     A   DISBUF
1017 BD 71 5E                       JSR         DISKEY
101A B7 12 01                       STA     A   TEMP2
101D BD 72 7D                       JSR         BINSEG
1020 97 11                          STA     A   DISBUF+1
1022 CE 12 00                       LDX         #TEMP1  POINT TO TEMP1
1025 BD 72 4D                       JSR         BYTE    FORM A BYTE
1028 B7 12 00                       STA     A   TEMP1
102B BD 71 5E                       JSR         DISKEY
102E B7 12 02                       STA     A   TEMP3
1031 BD 72 7D                       JSR         BINSEG
1034 97 15                          STA     A   DISBUF+5
1036 BD 71 5E               AGAIN   JSR         DISKEY
1039 B7 12 03                       STA     A   TEMP4
103C 27 08                          BEQ         PZERO
103E 7C 12 02                       INC         TEMP3   M=M+
1041 B1 12 02                       CMP     A   TEMP3   P=(M?
1044 2B 18                          BMI         NEWN
1046 CE 00 10               PZERO   LDX         #DISBUF
1049 86 06                          LDA     A   #6      'E'
104B 97 12                          STA     A   DISBUF+2
104D 86 15                          LDA     A   #$15    'R'
104F 97 13                          STA     A   DISBUF+3
1051 8D 61                          BSR         DISOUT
1053 CE 00 10                       LDX         #DISBUF POINT OF DISBUF
1056 86 7F                          LDA     A   #$7F    ' '
1058 A7 02                          STA     A   2,X
105A A7 03                          STA     A   3,X
105C 20 D8                          BRA         AGAIN
105E B6 12 00               NEWN    LDA     A   TEMP1   N
1061 B0 12 03                       SUB     A   TEMP4   N-P
1064 B7 12 00                       STA     A   TEMP1   SAVE NEW N
1067 BD 72 6E                       JSR         TWODIG
106A D7 10                          STA     B   DISBUF  MSN
106C 97 11                          STA     A   DISBUF+1 LSN
106E 8D 44                          BSR         DISOUT
1070 B6 12 00                       LDA     A   TEMP1   N
1073 4A                             DEC     A
1074 27 3C                          BEQ         CHEAT
                            *
                            **      CALCULATE NEW VALUES
                            *
1076 7F 12 05                       CLR         TEMP6   A
1079 7C 12 02                       INC         TEMP3   M=M+1
107C 7C 12 05               INCA    INC         TEMP6   A=A+1
107F B0 12 02                       SUB     B   TEMP3   (N-1)-(M+1)
1082 2E F8                          BGT         INCA
1084 7A 12 05                       DEC         TEMP6   A=A-1
1087 5F                             CLR     B
1088 FB 12 02               ADDM1   ADD     B   TEMP3   +(M+1)
108B 7A 12 05                       DEC         TEMP6   A=A-1
108E 26 F8                          BNE         ADDM1
1090 B6 12 00                       LDA     A   TEMP1   N
1093 10                             SBA                 N-A(M+1)
1094 16                             TAB
1095 5A                             DEC     B           R=N-A(M+1)-1
1096 26 02                          BNE         MISSET  R=0?
1098 C6 01                          LDA     B   #1      SET R=1
109A B6 12 00               MISSET  LDA     A   TEMP1   N
109D 10                             SBA                 N=N-R
109E B7 12 00                       STA     A   TEMP1   N
10A1 BD 72 6E                       JSR         TWODIG
10A4 D7 10                          STA     B   DISBUF
10A6 97 11                          STA     A   DISBUF+1
10A8 B6 12 00                       LDA     A   TEMP1
10AB 81 01                          CMP     A   #1      N=1?
10AD 27 04                          BEQ         CLOT
10AF 7E 10 36                       JMP         AGAIN
10B2 3F                     CHEAT   SW1
10B3 3F                     CLOT    SW1
                            *
                            **      DISOUT, DISPLAY ROUTINE WITH TIME-OUT
                            *
10B4 86 80                  DISOUT  LDA     A   #$80
10B6 B7 12 04                       STA     A   TEMP5   SET DISPLAY TIME
10B9 BD 71 DD               DISOU1  JSR         DISPLY
10BC 7A 12 04                       DEC         TEMP5
10BF 26 F8                          BNE         DISOU1
10C1 39                             RTS
                            *
                            **      TEMP STORE
1200                        TEMP1   RMB         1
1201                        TEMP2   RMB         1
1202                        TEMP3   RMB         1
1203                        TEMP4   RMB         1
1204                        TEMP5   RMB         1
1205                        TEMP6   RMB         1
                            *
```

This program simulates the game of "Pick up sticks" or "Matches". The winner is the player (either DATUM or the operator) who forces the opponent take the last object.

Table 1: DATUM Monitor routines

NAME	START LOCATION	COMMENTS
SPROMPT	7107	Puts a "PROMPT" in the left-most digit and blanks the rest.
DISKEY	715E	Refreshes the display then scans the keyboard. If no key was pressed then the routine loops back to display refresh routine.
MPXK	7177	Tests whether a key has been depressed. Acc B = 0 if no key depressed else # 0. Acc A has value of key. Index Reg is used.
TIMLP	71D2	Gives a 10ms delay and does not change Acc A, Acc B or Index Reg.
XTIMLP	71D7	Time delay depends on the Index Reg. Delay = Index Reg contents x8μS.
DISPLAY	71DD	Displays for 10msec contents of display locations 0010—0015. Index Reg used.
BYTE	724D	Takes two nibbles in successive memory locations (indexed) and combines to form a byte. MSN in location 00 and LSN location 01. Index Register must be set before entering subroutine. Result in Acc.
TWODIG	726E	Takes byte in Acc A and splits into 2 nibbles MSN in Acc B, LSN in Acc A Index Register.
BIN7SEG	727D	Converts a hex value in Acc A, into a segment code. This result is in Acc A, the original hex value being lost. Index Reg is left unchanged.
ADBX	734C	Adds the contents of Acc B to the Index Reg and the result is in the Index Reg on exit from the routine.
SBBX	7364	Is similar to ADBX but subtracts Acc B from the Index Register.

Table 2: Addresses of PIA 1 (hex).

6000 Data register A and Data Direction register A
6001 Control register A
6002 Data register B and Data Direction register B
6003 Control register B

Table 1 shows Monitor routines which can be used with any program. Table 2 has the addresses of PIA 1 of DATUM.

namely DISKEY, BYTE, TWODIG and BINTSEG. The function of each of these is explained in Table 1. Please note that MSN stands for most significant nibble and LSN for least significant nibble.

Concluding remarks

The above three programs illustrate some of the many applications of DATUM, and we hope that you have enjoyed studying and using them. The applications and usefulness of DATUM can be extended by adding the other two integrated circuits (the ACIA and PIA2), but even more, by adding the matching extension board. This board can be expanded according to user needs, and provides for memory to be increased up to 12k bytes (RAM and/or EPROM), analog to digital and digital to analog conversion, cassette interface and use with a terminal.

DATUM was originated by Malcolm Haskard, Senior Lecturer at the School of Electronic Engineering of The South Australian Institute of Technology, who designed the circuit and wrote the documentation. John Duval, a technician with the School was responsible for the artwork, including the printed circuit patterns used by Gammatron to produce the boards. As mentioned in the first article, Peter O'Neill, a masters degree student, wrote the DATUM monitor program.

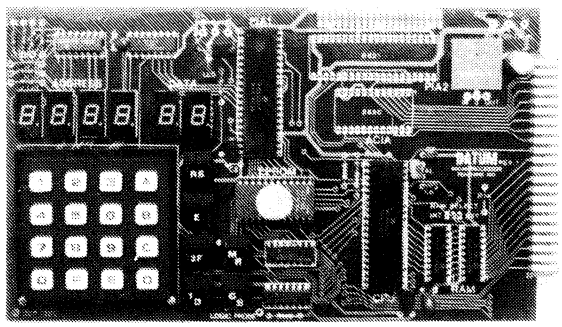
Complete kits and instructions for building the DATUM microprocessor board are available from Gammatron, Unit 1, Weens Rd, Pooraka, SA, 5095. Phone (08) 262 6555.

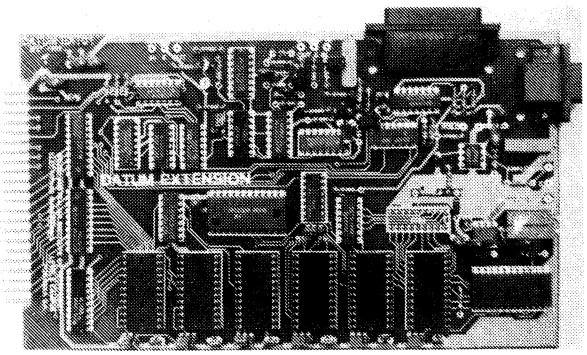
Correction

The connection diagram for the DATUM expansion interface, Fig. 8, published in November 1982, is partially incorrect. On the upper side of the board, starting from pin 11, the address lines are in the sequence A0, A1, A2, A3, A7, A4, A8, A5, A9, A6, A10, and A11 (on pin 22).

DATUM : THE COMPUTER KIT

(AS FEATURED IN ELECTRONICS AUSTRALIA, NOV., 82, DEC., 82 AND JAN., 83 ARTICLES)





DATUM is a minimum cost, self contained microcomputer kit designed by the South Australian Institute of Technology to assist in the teaching of the basics of microprocessor systems. The **DATUM** microcomputer is widely used as a process controller in a vast range of applications.

Current users include: Department of Defence; Telecom; Department of Aviation and various Technical Colleges etc.

The DATUM microcomputer incorporates:

DATUM Computer Kit $119.00
Additional PIA $4.80
Additional ACIA $4.80

DATUM Extension Kit (Basic) . . $119.00
(Including cassette interface, memory expansion and line buffers)

Options: A to D Conversion $21.00
D to A Conversion $29.00
RS232C Terminal $24.00

Manuals "Working with DATUM" Book 1 $13.00 (No S.T.)
Book 2 (Available September)

All prices include S.T., post and packing charge $5.00
Send money order, cheque, Bankcard authority, and we will post haste a kit.
DATUM is available exclusively from Gammatron Pty. Ltd.
Office:— Unit 1, Ween Road, Pooraka 5095
Postal:— P.O. Box 62, Ingle Farm, S.A. 5098

ASK ABOUT OUR LARGE RANGE OF COMPONENTS AT COMPETITIVE PRICES. WE ARE OPEN 7 DAYS A WEEK