Mk 14

Micro Computer Training Manual

Contents

Part 1

Part 2

Devised and written by: David Johnson—Davies
except programs marked thus*

1 Introduction to the kit

The MK14 comprises a full set of components to build up a completely functional computer.

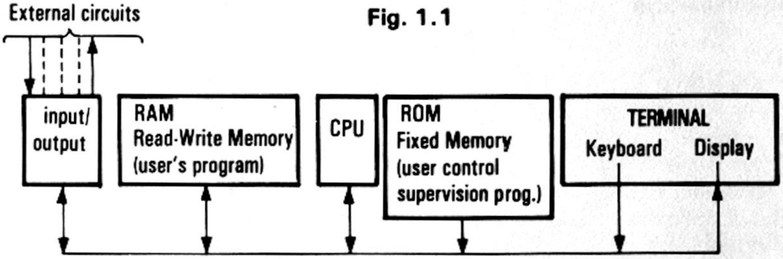
When the unit has been correctly assembled only the connection of a suitable power source is needed for the display to light up and the user then finds that command and control of the unit is literally at his fingertips via the keyboard.

Having mastered the simple rules for operation of the keyboard and interpretation of the display, it is immediately possible to study the workings of the system and the computer's instructions, and experiment with elementary programming.

From this point the user can progress to the library of ready-written programs available in Part II of this manual, and to programs of his own invention. Because of the inherently enormous versatility of the digital computer it is hard to suggest any particular direction which the independent programmer may take. Arithmetic, logic, time measurement, complex decision making, learning ability, storage of data, receiving signals from other equipment and generating responses and stimuli can all be called upon.

Thus calculators, games, timers, controllers (domestic, laboratory, industrial), or combinations of these are all within the scope of the machine.

Fig. 1.1



Components of the kit include central processor, pre-programmed control memory, read-write memory, input/output circuits, the terminal section i.e. the keyboard and display, and interfacing to the terminal.

This line-up corresponds to the basic elements present in even the most sophisticated multi-million pound computer. Indeed the fundamental principles are identical. However, the user of the MK14 who wishes to understand and utilise these principles has the advantage of being able to follow in detail the action and interaction of the constituent parts, which are normally inaccessible and invisible to the big computer operator.

Do not regard the MK14 as an electronics construction project. The MK14 is a computer, and computers are about software. It is the program which brings the computer to life, and it is the program which is capable of virtually infinite variation, adjustment and expansion.

Of course an understanding of the architecture of the machine and the functions of the separate integrated circuits is valuable to the user. But these aspects conform to a fairly standard pattern and the same straightforward set of interconnection rules regardless of the task or function the computer is performing.

2 The Manual -its objectives and uses

The MK14 is intended to bring practical computing to the widest possible range of users by achieving an absolute minimum cost. The wider the user spectrum, the wider, to be expected will be the variation of expertise the manual has to cater for; from the total novice, who wishes to learn the basic principles and requires thorough explanation of every aspect, to the experienced engineer who has immediate practical applications in view. Additionally, the needs of the beginner can be sub-divided into three parts:-

1. An informal step by step procedure to familiarise with the operation of the MK14. If this is arranged as an interactive 'do' and 'observe' sequence, it becomes a comparatively painless method of getting a practical 'feel' for the computing process. Section 5.
2. A formal definition/description of the significant details of the microprocessor itself, i.e. its architecture and instruction set. Users of all levels are strongly recommended to study this section, (Section 9) at an early stage. It is supported by a programme of practical exercises aimed to precisely demonstrate the elemental functions of the device, and the framework inside which they operate. It is emphasised that to gain the most complete fluency in what are the basics of the whole subject is not merely well worth the effort but is essential to the user's convenience?
3. An explanation of the general principles of the digital processor, along with the associated notation and conventions, section 6. This also breaks down into the joint aspects of hardware and software.

Clearly parts of the above will also prove useful to the knowledgable user who, however, will probably be able to skip the advice in section 3 on basic electronic assembly technique. The control part of this section contains information specifically pertinent to the MK14 and should be read by all.

Further sections to be referenced when the MK14 has been assembled, and the user has built up a working understanding, are those discussing programming techniques and methodology. From that point the applications examples of varying degrees of complexity and function, in Part II, should be possible for the reader to tackle.

Note: This manual applies to kits with Issue 4 or 5 boards and the revised SCIOS monitor.

3 Construction procedure Notes on soldering

The construction of the unit is a straightforward procedure consisting of inserting the components in the correct positions and soldering them in place. If this is done without error the system should become functional as soon as power is applied. To ensure that this happens without any hitches some recommendations and advice are offered. A step-by-step construction procedure with a diagram is laid down. An appendix to this section contains notes on soldering techniques.

Plug in socket option for integrated circuits

The I.C. components utilised in the MK14 are both robust and reliable. But accidents are possible—and should an I.C. be damaged either during construction or later, its identification and replacement is made many orders easier if devices are mounted in sockets. Socket usage is therefore most strongly recommended, particularly where the user is concerned with computing rather than electronics. Science of Cambridge offer a MK14 rectification service specifying a component cost only replacement charge when the system in question is socket equipped.

Integrated Circuit Device Handling

M.O.S. integrated circuits historically have gained a reputation for extreme vulnerability to damage from static electricity. Modern devices while not unbreakable embody a high degree of protection. This means that high static voltages will do no harm as long as the total energy dissipated is small and a practical rule of thumb is that if the environment is such that you yourself don't notice static shocks, neither will the I.C.

It is essential for the soldering iron to be earthed if I.C.'s are being soldered directly into the P.C. board. The earth must ground the soldering iron bit. This warning applies to any work carried out which might bring the soldering iron into contact with any I.C. pin.

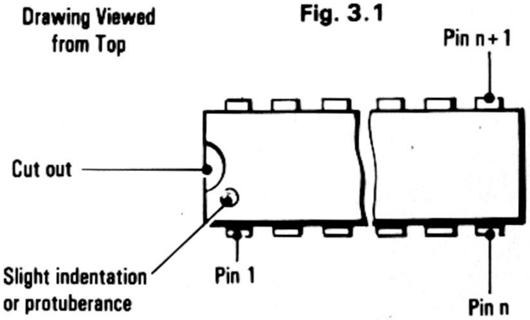
Catastrophe is achievable with minimum trouble if certain components are fitted the wrong way round.

Component Orientation and I.C. Pin Numbering

Three types belonging to the kit must be oriented correctly. These are the I.C.'s, the electrolytic capacitors and the regulator.

(i) I.C's are oriented in relation to pin 1. Pin 1 can be identified by various means; fig. 3.1 illustrates some of these:-

Fig. 3.1



Pin 1 itself may bear a faint indentation or a slight difference from other pins. The remaining pins are numbered consecutively anti-clockwise from Pin 1 viewing device as in Fig. 3.1.

Note position of type no. is **not** a reliable guide.

(ii) Electrolytic capacitors have a positive and a negative terminal. The positive terminal is indicated by a' + ' sign on the printed circuit. The capacitor may show a ' + ' sign or a bar marking by the positive terminal. The negative is also differentiated from the positive by being connected to the body of the device while the positive appears to emerge from an insulator.

(iii) The regulator has a chamfered edge and is otherwise asymmetrical-refer to assembly diagram.

Assembly Procedure

Equipment required—soldering iron, solder, side-cutters or wire snippers.

Step No. **Operation**

1 Identify all resistors, bend leads according to diagram and place on layout diagram in appropriate positions.

2 Insert resistors into printed circuit and slightly bend leads at back of board so that resistors remain in place firmly against the P.C.

3 Solder resistors in place and cut surplus leads at back of printed circuit.

4 Re-check soldered joints and component positioning.

5 Identify all capacitors, bend leads according to diagram and place on layout diagram in appropriate positions.

6 Insert capacitors into printed circuit and slightly bend leads behind board so that capacitors remain in place firmly against the P.C.

7 Solder capacitors in place and cut surplus leads behind P.C.

8 Check soldered joints, component positions and orientation.

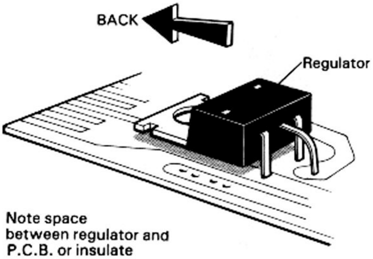
9 (If sockets are being used skip to step 14). Identify and place in position on diagram all I.C's with particular reference to orientation.

10 Insert I.C's into P.C. Note:- The I.C. pins will exhibit a degree of 'splay'. This allows the device to be retained in the P.C. mechanically after insertion so do not attempt to straighten, and use the following technique: place one line of pins so they just enter the board; using a suitable straight edged implement, press opposing row of pins until they enter the board; push component fully home.

11 Re-check device positioning and orientation with EXTREME care!

Step No.	Operation
12	Solder I.C's in place. It is not necessary to snip projecting pins.
13	Re-check all I.C. soldered joints. (skip to step 20)
14	Place appropriate sockets in position on diagram. See Fig. 3.2 (centre pages).
15	Insert first or next socket in P.C. board. These components are not self retaining so invert the board and press onto a suitably resilient surface to keep socket firmly against the board while soldering.
16	Solder socket into position.
	(repeat steps 14-16 until all sockets are fitted)
17	Identify and place into position on diagram all I.C's with particular reference to orientation.
18	Transfer I.C's one-by-one to P.C. assembly and place in appropriate sockets.
19	Check all socket soldered joints.
20	Insert regulator and solder into position. See Fig. 3.3.
21	Insert push button and solder into position. See Fig. 3.4.
22	Mount keyboard. See Fig. 3.5.
23	Mount display. See Fig. 3.6.
24	Ensure that all display interconnections are correctly aligned and inserted.
25	Solder display into position.
26	Solder crystal in position.
27	Re-check all soldering with special reference to dry joints and solder bridges as described in appendix on soldering technique.
28	(Optional but advisable). Forget the whole job for 24 hours.
29	Re-inspect the completed card by retracing the full assembly procedure and re-checking each aspect (component type, orientation and soldering) at each step. When the final inspection is satisfactorily completed proceed to section 4, Power Connect and Initial Operation.

Fig. 3.3



Note space
between regulator and
P.C.B. or insulate

Fig. 3.4

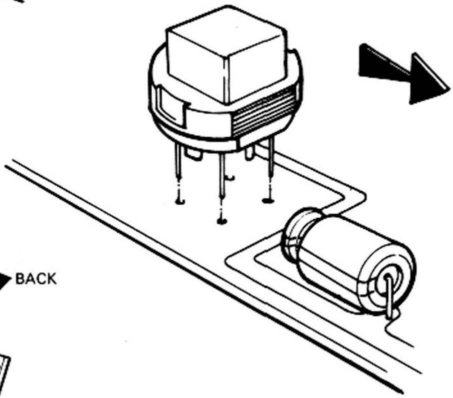


Fig. 3.6

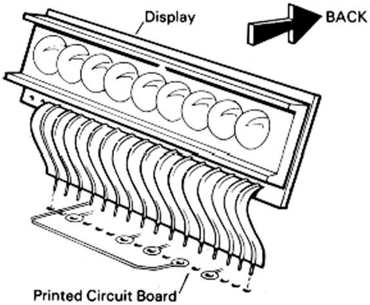
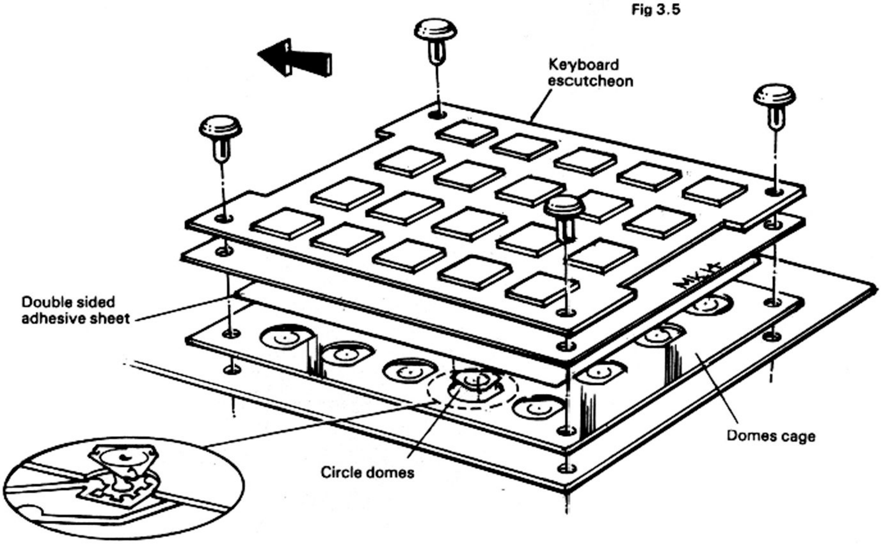


Fig 3.5



Appendix Soldering Technique

Poor soldering in the assembly of the MK14 could create severe difficulties for the constructor so here are a few notes on the essentials of the skill.

The Soldering Iron Ideally, for this job, a 15W/25W instrument should be used, with a bit tip small enough to place against any device pin and the printed circuit without fouling adjacent joints. IMPORTANT—ensure that the bit is earthed.

Solder resin cored should be used. Approx. 18 S.W.G. is most convenient.

Using the Iron The bit should be kept clean and be sufficiently hot to form good joints.
A plated type of bit can be cleaned in use by wiping on the dampened sponge (if available), or a damp cloth. A plain copper bit corrodes fairly rapidly in use and a clean flat working face can be maintained using an old file.A practical test for both cleanness and temperature is to apply a touch of solder to the bit, and observe that the solder melts instantly and runs freely, coating the working face.

Forming the Soldered Joint—with the bit thus 'wetted' place it into firm contact with **both** the component terminal and the printed circuit 'pad', being soldered together. Both parts must be adequately heated. Immediately apply solder to the face of the bit next to the joint. Solder should flow freely around the terminal and over the printed circuit pad. Withdraw the iron from the board in a perpendicular direction.
Take care not to 'swamp' the joint, a momentary touch with the solder should be sufficient. The whole process should be complete in one or two seconds. The freely flowing solder will distribute heat to all part of the joint to ensure a sound amalgam between solder and pad, and solder and terminal. Do not hold the bit against the joint for more than a few seconds either printed circuit track or the component can be damaged by excessive heat.

Checking the Joint A good joint will appear clean and bright, and the solder will have spread up the terminal and over the pad to a radius of about $\frac{1}{16}$ inch forming a profile as in Fig. 3.7(a).

Fig. 3.7

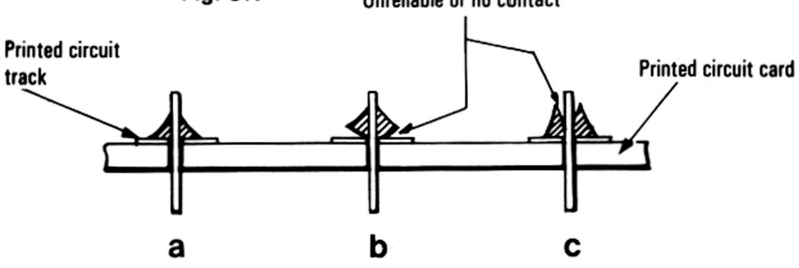


Fig 3.7 (b) and (c) show exaggerated profiles of unsuccessful joints. These can be caused by inadequate heating of one part, or the other, of the joint, due to the iron being too cool, or not having been in direct contact with both parts; or to the process being performed too quickly. An alternative cause might be contamination of the unsoldered surface.

Re-making the Joint Place the 'wetted' iron against the unsatisfactory joint, the solder will then be mostly drawn off. Re-solder the joint. If contamination is the problem it will usually be eliminated after further applications by the flux incorporated within the solder.

Solder 'Bridges'—can be formed between adjacent tracks on the printed circuit in various ways:—

(i) too cool an iron allowing the molten solder to be slightly tacky
(ii) excessive solder applied to the joint
(iii) bit moved away from the joint near the surface of the board instead of directly upwards

These bridges are sometimes extremely fine and hard to detect, but are easily removed by the tip of the cleaned soldering iron bit.

Solder Splashes—can also cause unwanted short circuits. Careless shaking of excess solder from the bit, or allowing a globule of solder to accumulate on the bit, must be avoided. Splashes are easily removed with the iron.

In summary, soldering is a minor manual skill which requires a little practise to develop. Adherence to the above notes will help a satisfactory result to be achieved.

4 Power Connect and Switch On

The MK14 operates from a 5V stabilised supply. The unit incorporates its own regulator, so the user has to provide a power source meeting the following requirements:—

Current consumption	Basic kit only: 400mA
	+RAM I/O option: +50mA
	+extra RAM option: +30mA

Max I/P permitted voltage (including ripple) 35V
Min I/P permitted voltage (including ripple) 7V

Batteries or a mains driven power supply may be used. When using unregulated supplies ensure that ripple at the rated current does not exceed the I/P voltage limits.

If a power source having a mean output voltage greater than IOV has to be used, a heat sink must be fitted to the regulator. A piece of aluminium or copper, approx. 18 s.w.g., of about two square inches in area, bolted to the lug of the regulator should permit input voltages up to about 18V to be employed.

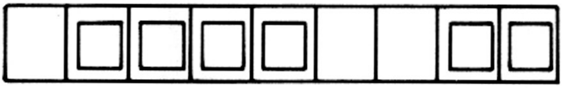
Alternatively a suitable resistor fitted in series with the supply can be used. To do this the value of the series resistor may be calculated as follows:-

2 × (minimum value I/P voltage -7) Ω
Resistor dissipation will be 0.5W/Ω

Having selected a suitable power supply the most important precaution to observe is that of correct polarity. Connect power supply positive to regulator I/P and power supply negative to system ground.

Switch on.

Proper operation is indicated by the display showing this:—



The left-hand group of four digits is called the address field and the right-hand group is the data field.

If this works—congratulations!

Before proceeding to the next section on operating the MK14 you should first tie the SENSE-A input to ground (unless you are using the single-step circuit). This involves linking pin 5 to pin 26 on the top edge connector.

5 Monitor Operation

To help the user become accustomed to commanding and interrogating the MK14 an exercise consisting basically of a sequence of keyboard actions, with the expected display results, and an explanatory comment, is provided.
Readers who are not familiar with hexadecimal notation and data representation should refer to section 7.
It will be clear to those who have perused the section dealing with MK14 basic principles that to be able to utilise and understand the unit it is necessary firstly to have the facility to look at the contents of locations in memory I/O and registers in the CPU, and secondly to have the facility to change that information content if desired.

The following shows how the monitor program held in fixed memory enables this to be done.

Examining Memory

Operator Action	Display	Comment
Reset	0000 00	The contents of memory location zero are are displayed in the data field.
MEM	0001 CF	Next address in sequence is displayed, and the data at that address.
MEM	0002 FF	Address again incremented by one, and the data at the new address is displayed.
MEM	0003 90	Next address and contents are displayed

The user is actually accessing the beginning of the monitor program itself. The items of data 00, CF, FF, 90 are the first four instructions in the monitor program.
It is suggested that for practise a list of twenty or thirty of these is made out and the appropriate instruction mnemonics be filled in against them from the list of instructions in Section 9. Additionally, this memory scanning procedure offers an introduction to the hexadecimal numbering method used by the addressing system, as each MEM depression adds one to the address field display.

Loading Memory

Note: symbol X indicates when digit value is unpredictable or unimportant.

Operator Action	Display	Comment
0	0000 XX	First digit is entered to address field, higher digits become zero.
F	000F XX	Second address digit keyed enters display from right.
1	00F1 XX	Third address digit keyed enters display from right.
2	0F12 XX	This is first address in RAM available to the user (basic version of kit).
TERM	0F12 XX	TERM changes to 'data-entry' mode.
1	0F12 01	Data placed in RAM.
MEM	0F13 XX	Address is incremented.
1	0F13 01	New data.
1	0F13 11	is keyed and placed in RAM.
MEM	0F14 XX	Data
22	0F14 22	is loaded
MEM	0F15 XX	into successive
33	0F15 33	locations
MEM	0F16 XX	

Operator Action	Display	Comment
ABORT	0F16 44	Get back into 'address-entry' mode.
0F12	0F12 01	Enter original memory address and
MEM	0F13 11	check that data
MEM	0F14 22	remains as
MEM	0F15 33	was
MEM	0F16 44	loaded.

Switch power off and on again. Re-check contents of above locations. Note that loss of power destroys read-write memory contents.
Repeat power off/on and re-check same locations several times—it is expected that RAM contents will be predominately zero, and tend to switch on in same condition each time. This effect is not reliable.

Entering and Executing a Program

Operator Action	Display	Comment
Reset	0000 00	
0, F, 2, 0	0F20 XX	Enter program starting at 0F20
TERM	0F20 XX	
9, 0	0F20 90	
MEM	0F21 XX	
F, E	0F21 FE	
ABORT	0F21 FE	
0, F, 2, 0	0F20 90	Enter start address
GO	blank	commence execution

The program consists of one instruction JMP-2 (90FE in machine code). 90 represents the jump operation; FE represents -2, meaning back two locations.
If this exercise failed to work you may have omitted to join the SENSE-A input to ground. See Section 4 again.

We have created the most elementary possible program—one that loops round itself. There is only one escape—RESET which will force the CPU to return to location 0.

RESET	0000 00	Reset does not affect memory the instruction JMP—2 is still lurking to trap the user.

Monitor Operation Summary

TERM	—Change to 'data entry' mode
MEM	—Increment memory address
ABORT	—Change to 'address entry' mode
GO	—Execute program at address displayed

Note that the valuedisplayed in the data field is the actual data at the location addressed, so an attempt to enter data into ROM will have no effect.

MK14 Schizophrenia

The exercises above illustrate a fundamental aspect of the MK14. While the user is entering commands and data through the keyboard and observing responses on the display the CPU is executing the monitor program which resides in the fixed data memory area. This is so, notwithstanding the fact that data values may be read and written in other parts of memory. All instructions are being derived from the monitor.

However as soon as GO command is entered, in conjunction with an address elsewhere in memory, the CPU is made to execute alternative program, and the characteristics of the system can be totally transformed. Thus an MK14 can have as many personalities as it can have different program in memory.

When in user program the MK14 is utterly ''unaware'' of the existence of the monitor, but the monitor does ''know'' certain basic data concerning the user program and the only bridge between monitor and user program is the GO command.

Specifying Register Contents

On transferring control from the monitor to your own program the CPU registers (except P3) are loaded from the block of read/write memory locations OFF9-OFFF. By modifying the contents of these locations you can specify the initial contents of the CPU registers before your program is executed.

Pointer register P3 is not saved alongside the other registers since it is used to hold the address for returning to the monitor program.

OFF9	P1H
OFFA	P1L
OFFB	P2H
OFFC	P2L
OFFD	A
OFFE	E
OFFF	S

Fig. 5.1 Image of CPU Registers in Read/Write Memory.

Inserting Breakpoints

The instruction:

X'3F (mnemonic XPPC 3).

if encountered in a program will cause an orderly return to the monitor. When control is returned to the monitor in this way the current contents of the SC/MP registers are copied out into the block of read/write memory OFF9-OFFF. Using the monitor 'memory examine' function you can then inspect this image of the registers and find out what they contained when the X'3F instruction was encountered.

Single-Step

The single-step facility allows you to step through a program being debugged, executing it a single instruction at a time, the next address and op-code being displayed after each step.

Some additional hardware is needed to implement the single-step facility; this is contained in two TTL packages, a 7493A binary counter, and a 7400 quad two-input NAND gate package. The circuit is shown in Fig. 5.1.

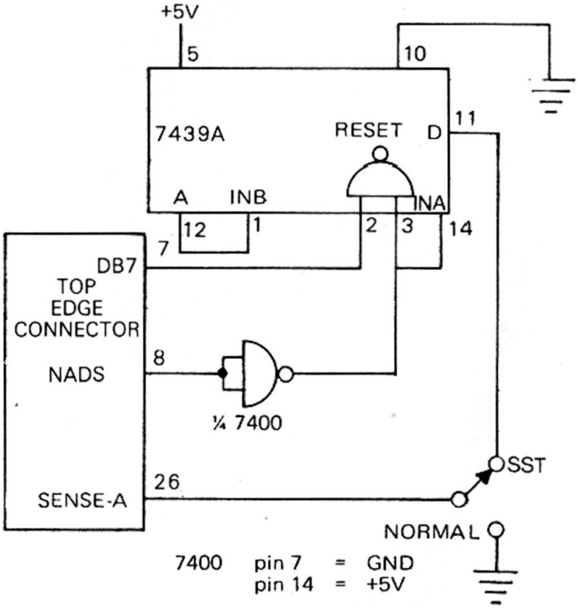


Fig. 5.1. Circuitry needed to implement single-step.

Note: the 7493 is also suitable but has a different pin-out.

Operation is as follows:

With the switch in 'NORMAL' position the monitor behaves as described previously. With the switch in 'SS' position pressing 'GO' will go to the program at the address displayed, but will only cause one instruction to be executed — the display will then show the next instruction and address. Repeatedly pressing 'GO' will step through the program. In between steps the contents of the user's registers, stored in RAM at OFF9 – OFFF, can be examined or altered using the monitor in the usual way. The switch may be returned to 'NORMAL' at any time; the next time 'GO' is pressed, the program will be executed in the normal way.

The single-step facility will appear to step over XPPC 3 instructions when encountered (X'3F) since they reverse the effect of the interrupt used for single-stepping. The single-step program will also behave strangely when used to step over HALT instructions (X'00), since the HALT flag is used by the single-step program.

Offset Calculation

The offset-calculation program is located at 0093 in the monitor, and it saves you the trouble of calculating jump operands.

Suppose we had the following program . . . .

```
ØFC9        C4ØØ        LOOP:       LDI Ø
             .                        .
             .                        .
             .                        .
             .                        .
             .                        .
             .                        .
ØFD4        9CXX                    JNZ LOOP
```

Where XX is to be determined. We use the offset program as follows:

1. Put the jump instruction address in (ØFF9, ØFFA), in this case ØFD4.
2. Put the destination address in (ØFFB, ØFFC), in this case ØFC9.
3. Enter the address ØØ93 and press 'GO'.
4. The program will insert the correct value of XX into your program, in this case F3.

Tape Interface Routines

The tape routines form a simple system for storing programs or data on a tape recorder and then reloading them anywhere in RAM memory. Programs can be relocated if they use suitable addressing modes.

The 'store to tape' program at 0052 converts a program or data to a series of long and short pulses. The 'load from tape' program at 007C reads back pulses in the same format and converts them back to the original binary data.

To convert these pulses to audio-frequency tone-bursts suitable for recording on a tape, and to convert the tone bursts back to pulses, the Science of Cambridge Cassette Interface Module can be used—available separately. This is shown in block form in Fig. 5.2.

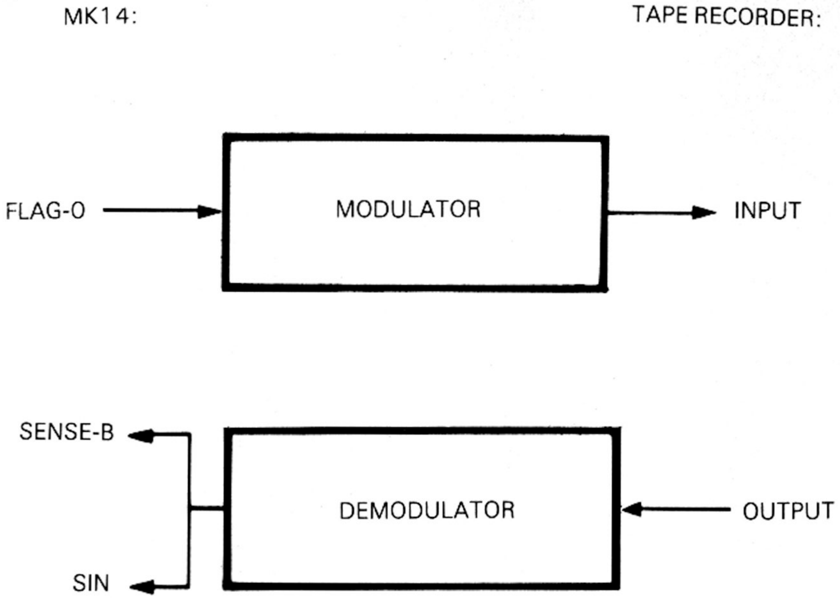


Fig. 5.2. Block Diagram of Cassette Interface Module.

Data Along a Serial Line

The tape interface routines can be used to transmit data or programs along a serial line between two MK14s; see Fig. 5.3.

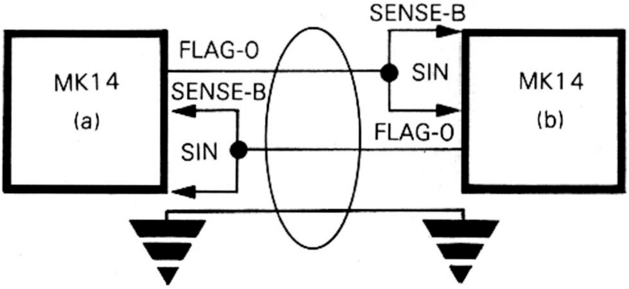


Fig. 5.3. Two MK14s communicating along a Serial Line.

Operation of the tape routines:

Writing to tape:

1. Put the number of bytes to be stored in location 0FF8.

   This will be a hex number not exceeding 256 . If one wanted to store seventy bytes one would enter 46.

   (NB: to store the full 256 bytes one should enter 00).
2. Put the starting address of the program to be stored in 0FF9 and 0FFA (P1H and P1L).
3. Go to 0052. The program will return to the monitor when the data has been transmitted.

Reloading from tape:

1. Put the starting address of where the program on tape is to be reloaded in OFF9 and OFFA (P1H and P1L).
2. Go to 007C. The program should be stopped with Reset when all the data has been read.

Monitor Subroutines

There are also routines in the monitor which may be used by one's own programs—for more details see section 10 of the manual.

Addresses of Routines in the monitor

Offset calculation	0093
Store to tape	0052
Load from tape	007C
Make 4 digit address	011B
Data to segments	0140
Address to segments	015A
Display and keyboard input	0185

6 Basic Principles of the MK14

Essentially the MK14 operates on exactly the same principles as do all digital computers. The 'brain' of the MK14 is a SC/MP micro-processor, and therefore aspects of the SC/MP will be used to illustrate the following explanation. However the principles involved are equally valid for a huge machine from International Computers down to pocket calculators. Moreover, these principles can be stated quite briefly, and are essentially very simple.

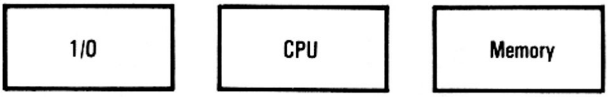
'Stored Program' Principle

The SC/MP CPU (Central Processing Unit) tends to be regarded as the centre-piece because it is the 'clever' component—and so it is. But by itself it can do nothing. The CPU shows its paces when it is given INSTRUCTIONS. It can obey a wide range of different orders and perform many complex digital operations. This sequence of instructions is termed the PROGRAM, and is STORED in the MEMORY element of the system. Since these instructions consist of manipulation and movement if data, in addition to telling the CPU what to do, the stored program contains information values for the CPU to work on, and tells the CPU where to get information, and where to put results.

Three Element System

By themselves the two fundamental elements CPU and MEMORY can perform wondrous things—all of which would be totally useless, since no information can be input from the outside world and no results can be returned to the user. Consequently a third element has to be incorporated —the INPUT/OUTPUT (I/O) section.

Fig. 6.1 The Three Element System



These three areas constitute the HARDWARE of the system, so called because however you may use or apply the MK14, these basic structures remain the same.

Independence of Software (stored Program) and Hardware

As with the other hardware, whatever particular instruction sequence is present within the memory at any one time, the basic structure of the memory element itself is unaltered.

It is this factor which gives the MK14 its great versatility: by connecting up its I/O and entering an appropriate program into its memory it can perform any digital function that can be contained within the memory and I/O size.

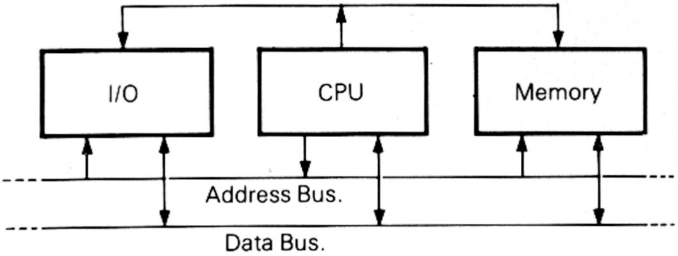
Random Access Memory (RAM)

Further, when the memory in question consists of a read **and write** element (RAM), in contrast to read **only** memory (ROM), this flexibility is enhanced, as program alterations, from minor modifications, to completely different functions, can be made with maximum convenience.

Interconnection of Basic Elements

Element inter-connection is standardised as are the elements themselves. Three basic signal paths, ADDRESS BUS (ABUS), DATA BUS (DBUS) and CONTROL BUS, are required.

Fig. 6.2 Interconnections of Three Element System



These buses are, of course, multi-line. In the MK14 the Abus = 12 lines, Dbus = 8 lines and Control bus = 3 lines. Expansion of memory or I/O simply requires connection of additional elements to this basic bus structure.

MK14 System Operation

Consider the MK14 with power on and the RESET signal applied to the SC/MP. This forces all data inside the CPU to zero and prevents CPU operation.

When the RESET is released the CPU will place the address of the first instruction on the Abus and indicate that an address is present by a signal on the ADDRESS STROBE (NADS) line which is within the control bus.

The memory will then respond by placing the first instruction on the Dbus.

The CPU accepts this information and signals a READ STROBE (NRDS) via a line within the control bus.

The CPU now examines this instruction which we will define as a no-operation, (instructions are normally referred to by abbreviations called MNEMONICS, the mnemonic for this one is NOP).

In obedience the CPU does nothing for one instruction period and then sends out the address of the second instruction. The memory duly responds with a Load Immediate (LDI). The CPU interprets this to mean that the information in the next position, in sequence, in memory will not be an instruction but an item of data which it must place into its own main register (ACCUMULATOR). so the CPU puts out the next address in sequence, and when the memory responds with data, then obeys the instruction.

The CPU now addresses the next position (LOCATION) in memory and fetches another instruction—store (ST). This will cause the CPU to place the data in the accumulator back on the Dbus and generate a WRITE STROBE (NWDS) via the control bus, (The program's intention here is to set output lines in the I/O element to a pre-determined value).

Before executing the store instruction the CPU addresses the next sequential location in memory, and fetches the data contained in it. The purpose of this data word is to provide addressing information needed, at this point, by the CPU.

So far, consecutive addresses have been generated by the CPU in order to fetch instructions or data from memory. In order to carry out the store instruction the CPU must generate a different address, with no particular relationship to the instruction address itself, i.e. an address in the I/O region.:

The CPU now constructs this address using the aforementioned data word and outputs it to the Abus. The I/O element recognises the address and accepts the data appearing on the Dbus (from the CPU accumulator), when signalled by the writer strobe (NWDS), also from the CPU.

Now the CPU reverts to consecutive addressing and seeks the next instruction from memory. This is an Exchange Accumulator with Extension register (XAE) and causes the CPU to simultaneously move the contents of the accumulator into the extension (E) register, and move the contents of the extension register into the accumulator. The programmer's intention in using this instruction here, could be to preserve a temporary record of the data recently written to the I/O location.

No new data or additional address information is called for, so no second fetch takes place. Instead the CPU proceeds to derive the next instruction in sequence.

For the sake of this illustration we will look at a type of instruction which is essential to the CPU's ability to exhibit intelligence.

This is the jump (JMP) instruction, and causes the CPU to depart from the sequential mode of memory accessing and 'jump' to some other location from which to continue program execution.

The JMP will be back to the first location.

A JMP instruction requires a second data word, known as the DISPLACEMENT to define the distance and direction of the jump.

Examining the memory I/O contents map, Fig. 6.3, shows location 0 to be seven places back from the JMP displacement which therefore must have a numerical value equivalent to—7. (Detail elsewhere in this manual will show that this value is not precisely correct, but it is valid as an example).

The instruction fetched after executing the JMP will be the NOP again.

In fact the sequence of five instructions will now be reiterated continually. The program has succumbed to a common bug—an endless loop, in which for the time being we will leave it.

Fig. 6.3 Map of Memory Location Contents.

LOCATION No.	LOCATION CONTENTS	
0	NOP (instruction)	MEMORY REGION
1	LDI (instruction)	
2	data (for use by LDI)	
3	ST (instruction)	
4	address information (for use by ST)	
5	XAE (instruction)	
6	JMP (instruction)	
7	−7 (displacement for JMP)	
Formed by CPU using data in loc. 4	Initially undefined—after 3 becomes same as loc. 2	I/O REGION

This brief review of a typical sequence of MK14 internal operations has emphasised several major points. All program control and data derives from the memory and I/O. All program execution is performed by the CPU which can generate an address to any location in memory and I/O, and can control data movement to or from memory and I/O.

Some instructions involve a single address cycle and are executed within the CPU entirely. Other instructions involve a second address cycle to fetch an item of data, and sometimes a third address cycle is also needed. For the sake of simplicity this outline has deliberately avoided any detail concerning the nature of the instruction/data, and the mechanics of the system. These subjects are dealt with in greater depth in sections 7 and 9.

7 MK14 Language-Binary and Hexadecimal

Discussion of the MK14 in this handbook so far has referred to various categories of data without specifying the physical nature of that data. This approach avoids the necessity of introducing too many possibly unfamiliar concepts at once while explaining other aspects of the workings of the system.

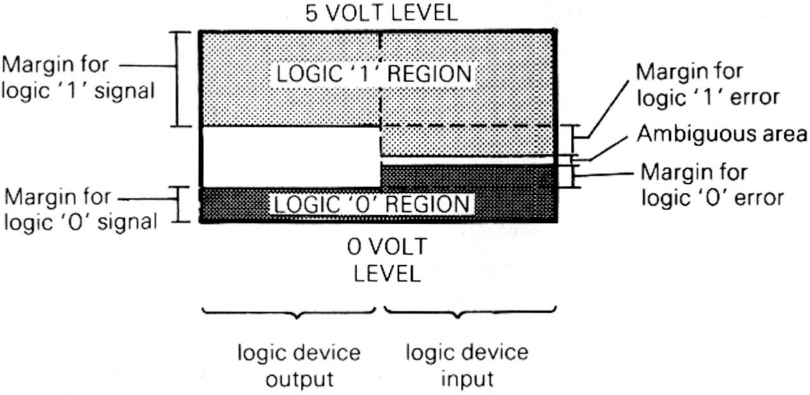
This section, then, gives electrical reality to the abstract forms of information such as address, data, etc., which the computer has to understand and deal with.

Binary Digit Computers use the most fundamental unit of information that exists—the binary digit or BIT—the bit is quite irreducible and fundamental. It has two values only, usually referred to as '0' and '1'. Human beings utilise a numbering system possessing ten digits and a vocabulary containing many thousands of words, but the computer depends on the basic bit.

However, the bit is readily convertible into an electrical signal. Five volts is by far the most widely used supply line standard for electronic logic systems. Thus a zero volt (ground) level represents '0', and a positive five volt level represents '1'. Note that the SC/MP CPU follows this convention which is known as positive logic; negative logic convention determines inverse conditions.

Logic Signal Voltage Limits For practical purposes margins must be provided on these signal levels to allow for logic device and system tolerances. Fig. 7.1 shows those margins.

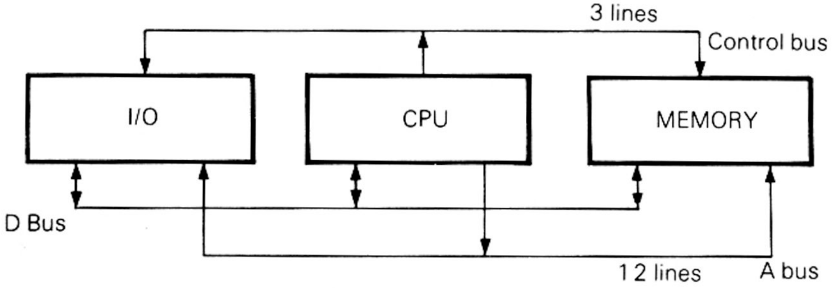
Fig. 7.1



'0's and '1's Terminology Many of the manipulation rules for '0's and '1's are rooted in philosophical logic, consequently terms like 'true' and 'false' are often used for logic signals, and a 'truth table' shows all combinations of logic values relating to a particular configuration. The control engineer may find 'on' and 'off' more appropriate to his application, while an electronic technician will speak of 'high' and 'low', and to a mathematician they can represent literally the numerals one and zero.

Using Bits in the MK14 The two state signal may appear far too limited for the complex operations of a computer, but consider again the basic three element system and it's communication bus.

Fig. 7.2



The data bus for example comprises eight lines. Using each line separately permits eight conditions to be signalled. However, eight lines possessing two states each, yield 256(2^8) combinations, and the A bus can yield 4096 combinations.

A group or WORD of eight bits is termed a BYTE

The Special 'K'

In computerese K is frequently used as a measure of memory size, or of the amount of memory space taken up by items of software. It is equal to 1024 (which is 2^{10}).

64K of memory actually contains over 65,000 locations, and is therefore sometimes termed 65K.

In relation to the SC/MP CPU and the MK14 the SC/MP has the capability to address a 64K (65536) memory, but the MK14 configuration utilises 4K (4096) out of this total. The monitor program occupies ½K (512) locations and the basic RAM and optional RAM each consist of ¼K (256) locations.

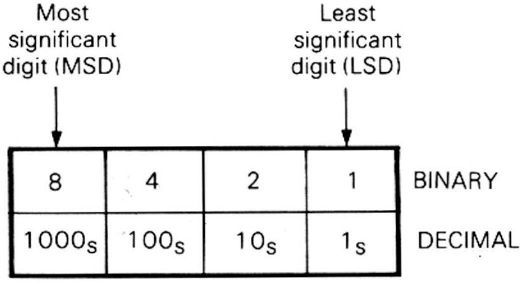
Decoding In order to tap the information potential implied by the use of combinations, the elements in the MK14 all possess the ability to DECODE bit combinations. Thus when the CPU generates an address, the memory I/O element is able to select one out of 4096 locations. Similarly, when the CPU fetches an instruction from memory it obeys one out of 128 possible orders.

Apart from instructions, depending on context, the CPU treats information on the data bus sometimes as a numerical value, or sometimes simply as an arbitrary bit pattern, thereby further expanding data bus information capacity.

Bits as Numbers When grouped into a WORD the humble bit is an excellent medium for expressing numerical quantities. A simple set of rules exist for basic arithmetic operations on binary numbers, which although they lead to statements such as 1 + 1 = 10, or 2_{10} and 2_{10} make 100_2, they can be executed easily by the ALU (Arithmetic and Logic Unit) within the CPU. Note that the subscripts indicate the base of the subscripted numbers.

Binary Numbers The table below compares the decimal values 0—15 with the equivalent binary notation.

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111



Place values in binary and decimal systems

Fig. 7.3

The binary pattern is self evident, and it can also be seen how place value of a binary number compares with that in the decimal system. Expressed in a different way, moving a binary number digit one place to the left doubles its value, while the same operation on a decimal digit multiplies its value by ten.

Binary Addition—requires the implementation of four rules:—

0 + 0 = 0
0 + 1 or 1 + 0 = 1
1 + 1 = 1 with carry (to next higher digit)
1 + 1 + carry (from next lower digit) = 1 with carry (to next higher digit)

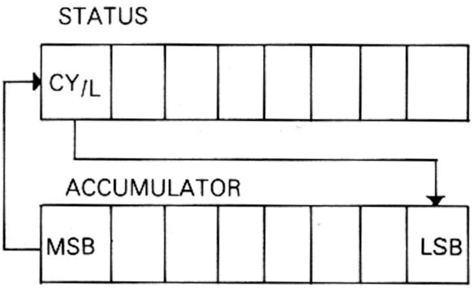
```
Example:—     1110110
             +1010101
             ________
             11001011
             111 1
             ↖↖↖ ↖  ←carry indications
```

Addition within the Computer

The working data unit of the SC/MP micro-processor is an eight bit word. The Arithmetic and Logic Unit (ALU) within the SC/MP can form the sum of two eight bit words according to the rules described above. The sum is always deposited in the Accumulator (AC) register following an arithmetic instruction.

Two eight bit numbers may generate a nine bit sum, so a ninth bit is provided. This is referred to as the Carry/Link bit, and is physically located in the leftmost position of the Status register. It has, however, no direct relation with any other part of the Status register which is really an assemblage of independent bits which may be moved as a unit for various purposes. The term 'link' in the bit name refers to a different function which will be explained elsewhere.

When the SC/MP executes an arithmetic instruction the initial condition of the Carry/Link bit is treated as a carry into the low order (least significant bit) of the sum. When the instruction is complete the Carry/Link bit assumes the value of the carry out from the most significant, or high order, bit of the sum, refer to Fig. 7.4.



LSB = Least Significant Bit
MSB = Most Significant Bit

Fig. 7.4. Status register and Inter-action CY/L bit with AC in Add Operations

There are two instructions available which control the CY/L bit apart from arithmetic and shift operations. They are Set Carry/Link (mnemonic = SCL), and Clear Carry/link (mnemonic = CCL) which set the Carry/Link to '1' and '0' respectively.

Binary Subtraction

0 — 0 = 0
1 — 1 = 0
1 — 0 = 1
0 — 1 = 1 with borrow (from next higher digit)
0 — 1 — borrow (from next lower digit) = 1 with borrow (from next higher digit)

```
Examples:—        0 1           0 1 1          0 101  ←—— borrow
                  1̸01           1̸0̸0            110          indications
                 −010          −001           −011
                  ‾‾‾           ‾‾‾            ‾‾‾
                  011           011            011
```

Subtraction by Addition in the Computer

The logic which would be required to implement the subtraction rules stated previously is not built into the ALU. Instead, subtraction is carried out using the add capability, in conjunction with an ability to form a logical complement. That is, to operate on a binary word such that '1's are changed to '0's, and '0's are changed to '1's.

Consider a four bit binary value. Fig. 7.5. Adding the complement of the value to the value itself will always generate all '1's, the further addition of 1 will generate all '0's, with a carry from the high order bit.

```
Value              1010
Complement         0101 +
                   1111
                      1 +
                  10000
```

Fig. 7.5.

Two's Complement

If we overlook the carry in the result, the complement of a binary value plus one can be regarded as a negative quantity of equivalent size, and is known as the Two's Complement.

The SC/MP CPU possesses a Complement and Add (mnemonic = CAD) instruction which therefore contributes to a subtract function. By utilising the property of CY/L bit as defined earlier, the requisite additional one can be provided.

If the program ensures that the CY/L bit is initially set, the CAD instruction will automatically generate a correct result.

A further difficulty remains. What is to be done when the net result of a subtraction is negative? The four bit word in the example offers no indication of sign. The answer rests with the CY bit. Fig. 7.6 shows that when the result is positive or zero the CY will be set, and when it is negative the CY will be clear.

```
a) 1010—1011            b) 1010—1011            c) 1010—1001


            1010                    1010                    1010
          +0101                   +0100                   +0110
          -----                   -----                   ------
            1111                    1110                  1 0000
          +    1                  +    1                    0001
          ------                  -----                   ------
          1 0000                   01111                  1 0001
```

Fig. 7.6 Subtraction Operations and State of the Carry

The behaviour of the CY bit lends itself equally conveniently to multiple precision subtraction, where it takes on the character of a borrow. Since a negative result is accompanied by a clear CY bit, a following CAD will yield a result smaller by one, thus implementing a borrow.

The principle remains valid when a subtraction is made from a negative value—the result is also a two's complement negative number.

Multiple Precision Arithmetic

This rather grandiose phrase simply means performing arithmetic on words larger than the CPU accumulator. An operation which is made easy by the dual function of the CY/L bit, ie the ability to represent the carry in, before an add instruction and the carry out, subsequently.

Clearly providing the programmer arranges for arithmetic operations on large words to commence with the lowest order eight bit word or byte, all carries arising between bytes will be correctly propagated.

Equally clearly, the programmer must ensure that the CY/L is initially in the appropriate state by use of the SCL or CCL instructions.

Hexadecimal Numbers

The ''0's'' and ''1's'' which make up binary words are unwieldy and long-winded to handle. Hexadecimal notation is a convenient means of compacting these binary words and is used widely in the context of micro-processor and digital systems in general.

Each group of four bits in a binary word is reduced to a single digit. Since there are sixteen combinations of four bits, sixteen characters are used; the ten decimal numerals and six alphabetic characters.

BINARY	HEXADECIMAL
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Fig. 7.7

Fig. 7.7 lists hexadecimal characters against the equivalent binary codes. To identify a hexadecimal value when written, the convention "X" is used as a prefix.
In some of the applications programs the alternative convention of prefixing the number with a zero is used; e.g. 07F (= X'7F).
Two examples of hexadecimal usage taken from the MK14 monitor program listing are given in Fig. 7.8.

i) X'018B Hexadecimal location address

0000 0001 1000 1101 Location address in binary

395 Location address in decimal derived from
1(16 × 16) + 8(16) + 11

ii) X'35 Hexadecimal location contents

0011 0101 Location contents in binary
XPAH 1 Equivalent instruction mnemonic

Fig. 7.8 Examples of the Hex Notation

B.C.D. Numbers

Binary Coded Decimal Notation is a means of expressing the ten decimal digits in binary code by directly converting each decimal digit into a four bit binary group. Fig. 7.9 compares decimal and B.C.D.
B.C.D. is really a sub-set of hexadecimal which discards the six highest order binary combinations. This convention is employed so as to retain the digits of a decimal number as separate entities, and to avoid converting the decimal value to pure binary.
The SC/MP CPU can perform an addition of two 2 digit B.C.D. words to generate a correct B.C.D. result. This is the DAD (Decimal add instruction).

An eight bit B.C.D. word can express the decimal values 0—99. The convenience that B.C.D. offers, i.e. that of allowing the programmer to use decimal data within the processor in the same format as the outside world data, has to be offset against the greater capacity (0—255_{10}) of the binary word.

Fig. 7.9

B.C.D.	**Decimal**
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

8 Program Notes

At the point the reader is likely to be considering the application programs in Part II and perhaps devising some software of his own. This section examines the manner in which a program is written and set out, the planning and preparation of a program, and some basic techniques.

When embarking on a program two main factors should be considered, they are: (i) hardware requirements, (ii) sequence plan.

Hardware Requirements An assessment should be made of the amount of memory required for the instruction part of the program, and the amount needed for data storage. In a dedicated micro-processor system these will occupy fixed, and read-write memory areas respectively. In the MK14, of course, all parts of the program will reside in read-write memory, simplifying the programmers task considerably, since local pools for data can be set up indiscriminately.

However, even in the MK14 more care must be given to the allocation of memory space for common groups of data and for input/output needs.

The SC/MP C.P.U. offers a certain amount of on-chip input/output in terms of three latched flags, two sense inputs, and the serial in/serial out terminals. So the designer must decide if these are more appropriate to his application than the memory mapped I/O available in the RAMIO option.

Memory Map A useful aid in this part of the process is the memory map diagram which gives a spatial representation to the memory and I/O resources the programmer has at his disposal. Fig. 8.1 shows the MK14 memory map including both add-in options

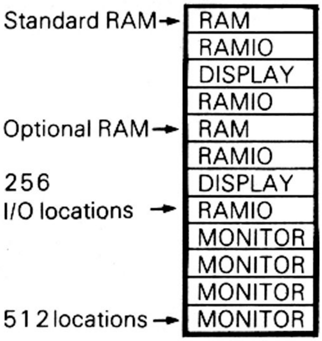


Fig. 8.1

The map displays the memory as a column of 4K locations, (in this case each of eight bits), with location zero at the base and addresses ascending upwards.

The reader may be surprised that various sections of memory appear to reside in several areas at once.

For example the monitor is repeated four times in the lower 2K block. Note also that the monitor will only operate correctly if executed in the lowest section, as only this section has the proper relationship to the RAM at the top.

These multiple appearances of memory blocks are due to partial address decoding technique employed to minimise decode components.

The map readily indicates that a CPU memory pointer (which can permit access to a block of 256 I/O locations) set to 0900_{16} would give the program a stepping stone into the display O/P or the RAMIO facilities.

Flow Chart The flow chart provides a graphical representation of the sequence plan. A processor is essentially a sequential machine and the flow chart enforces this discipline. Fig. 8.2 is a very simple example of a program to count 100 pulses appearing at an input. Three symbols are used (i) the **circle** for entry or exit points (ii) the **rectangle** for program operations (iii) the **diamond** for program decisions.

A flow chart should always be prepared when constructing a program. Each block is a convenient summary of what may be quite a large number of instructions. Of particular value is the overview provided of the paths arising from various combinations of branch decisions.

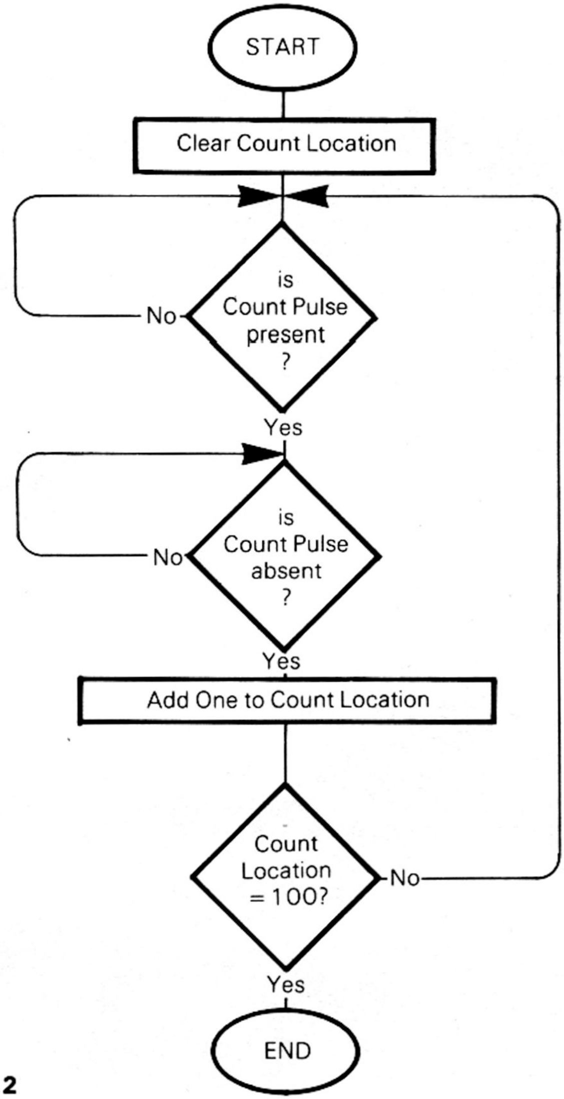


Fig. 8.2

The flow chart can reveal wasteful repetition or logical anomalies, and ensures that like a good story, the program starts at the beginning, progresses through the middle, and comes to a satisfactory end.

Program Listings

There is a well established convention and format for writing down a program listing which makes it much easier to understand than a list of the hex codes would be.

The application program listings at the end of this manual are given in this symbolic form known as 'assembler listings'.

We will examine a line from the monitor listing to define the various functions of the notation:

(a)

```
 (a)        (b)       (c)        (d)    (e)  (f) (g)      (h)
 0001       CFFF      INIT:      ST     @    —1 (3)   ;SO P3=-1
```

a) Location Counter. The current value of the location counter (program counter in the CPU) is shown wherever it is relevant e.g. when the line contains a program instruction or address label.

b) Machine Code. The actual code in the memory is shown here. As it is a two byte instruction the first two hexadecimal digits CF are in location 1 and FF is in location 2.

c) Symbolic Address Label. This is followed by a colon. Entry points to sub-sections of program can be labelled with meaningful abbreviations making the program easier to follow manually e.g. at some other place in the program a JUMP TO 'INIT' might occur. Automatic assemblers create an internal list of labels and calculate the jump distances.
However the MK14 user must do it by hand or with the offset-calculation program (see Section 5).

d) Mnemonic ST means STORE. The basic op-code for ST is C8.

e) The '@' symbol represents auto-indexed addressing; 4 is added to the basic op-code.

f) Displacement, or disp., in this case—1 or FF, forms the second byte of the instruction. Could alternatively be a symbolic name, in which case its value is calculated during assembly.

g) Pointer designation; specifies that P3 is to be referenced by this instruction. The pointer number is added to the basic op-code: C8 + 4 + 3 = CF (see d above).

h) Comment. All text following the semi-colon is explanatory material to explain the purpose of the instruction or part of programme.

Assembler conventions

In assembler listings the op-codes are represented by mnemonic names of from 2 to 4 letters, with the operands specified as shown:

```
LD disp              ;PC-relative addressing
LD disp (ptr)        ;Indexed addressing
LD @ disp (ptr)      ;Auto-indexed addressing
```

Constants and addresses are also sometimes represented by names of up to six letters; these names stand for the same value throughout the program, and are given that value either in an assignment statement, or by virtue of their appearing as a label to a line in the program. Some conventions used in these listings are shown below:

Directive statements

Assembler Format	Function
.END (address)	Signifies physical end of source program.
.BYTE exp (,exp...)	Generates 8-bit (single-byte) data in successive memory locations.
.DBYTE exp (,exp,...)	Generates 16-bit (double-byte) data in successive memory locations.

Assignment statements

LABEL:	SYMBOL = EXPRESSION	;Symbol is assigned ;value of expression
	. = 20	;Set location counter ;to 20
TABLE:	. = . + 10	;Reserve 10 locations for table

Loading Application Programs

These points will be illustrated with reference to the following short program.

```
                 ; LONG DELAYS
                 ;
                 ;
        00FF     TIME          =        X'FF
0000                           .=0F20
0F20             COUNT:        .=.+1
                 ;
0F21    8FFF     BEGIN:        DLY      TIME
0F23    A8FC                   ILD      COUNT
0F25    9CFA                   JNZ      BEGIN
0F27    3F                     XPPC     3

        0000                   .END
```

The leftmost column in the assembler listing is the address field; the next field is the data field. Where both a four-digit address and two digits of data are given, as in XPPC 3 above, the data shown should be entered into the memory at the specified address. When a four-digit address and four digits of data are given, as in DLY TIME above, the first two digits of data should be entered at the specified address, and the last two digits of data should be entered at the subsequent address.

There are two cases in the assembler listing where nothing needs to be entered into memory. The first is when only a data field is present, as in the assignment statement TIME = X'FF above. The value in the data field shows the value assigned to the symbol concerned (TIME in this case). The directive .END generates 0000 in the data field and simply specifies the end of the program. The second case is where only the address field is present; there are two examples in the above program. In the assignment statement . = 0F20 the location counter, symbolised by a dot, is set to the value of 0F20; its previous value, zero, is shown in the address field. If a label is attached to such an assignment statement, as in COUNT . = . + 1, the label gets the previous value of the location counter so that COUNT stands for the memory location 0F20 in this program.

BYTE Statements

In the programs 'Function Generator', 'Music Box', and 'Message' the values specified in the .BYTE directive statements should be loaded into successive memory locations starting at the address shown in the address field for each statement. In the 'Reaction Timer' the .DBYTE statements specify two bytes which should be loaded into the locations starting at the address shown in the address field.

9 Architecture and Instruction Set

The SC/MP microprocessor contains seven registers which are accessible to the programmer. The 8-bit accumulator, or AC, is used in all operations. In addition there is an 8-bit extension register, E, which can be used as the second operand in some instructions, as a temporary store, as the displacement for indexed addressing, or in serial input/output. The 8-bit status register holds an assortment of single-bit flags and inputs:

SC/MP Status Register

7	6	5	4	3	2	1	0
CY/L	OV	S_B	S_A	IE	F_2	F_1	F_0

Flags	**Description**
F_0-F_2	User assigned flags 0 through 2.
IE	Interrupt enable, cleared by interrupt.
S_A,S_B	Read-only sense inputs. If IE = 1, S_A is interrupt input.
OV	Overflow, set or reset by arithmetic operations.
CY/L	Carry/Link, set or reset by arithmetic operations or rotate with Link.

The program counter, or PC, is a 16-bit register which contains the address of the instruction being executed. Finally there are three 16-bit pointer registers, P1, P2, and P3, which are normally used to hold addresses. P3 doubles as an interrupt vector.

Addressing Memory

All memory addressing is specified relative to the PC or one of the pointer registers. Addressing relative to the pointer registers is called indexed addressing. The basic op-codes given in the tables below are for PC-relative addressing. To get the codes for indexed addressing the number of the pointer should be added to the code. The second byte of the instruction contains a displacement, or disp., which gets added to the value in the PC or pointer register to give the effective address, or EA, for the instruction. This disp. is treated as a signed twos-complement binary number, so that displacements of from -128_{10} to $+127_{10}$ can be obtained. Thus PC-relative addressing provides access to locations within about 128 bytes of the instruction; with indexed addressing any location in memory can be addressed.

Instruction Set

7 . . . 3	2	1 0
Op	m	ptr

byte 1

7 0
disp

byte 2

Memory Reference

Mnemonic	Description	Operation	Op Code Base
LD	Load	(AC)←(EA)	C000
ST	Store	(EA)←(AC)	C800
AND	AND	(AC)←(AC) A (EA)	D000
OR	OR	(AC)←(AC) V (EA)	D800
XOR	Exclusive-OR	(AC)←(AC) V (EA)	E000
DAD	Decimal Add	(AC)←$(AC)_{10}$ + $(EA)_{10}$ + (CY/L);(CY/L)	E800
ADD	Add	(AC)←(AC) + (EA) + (CY/L);(CY/L),(OV)	F000
CAD	Complement and Add	(AC)←(AC) + ¬(EA) + (CY/L);(CY/L),(OV)	F800

Base Code Modifier

Op Code = Base + m + ptr + disp

Address Mode	m	ptr	disp	Effective Address
PC-relative	0000	0000	00xx	EA = (PC) + disp
Indexed	0000	0100 0200 0300	00xx	EA = (ptr) + disp
Auto-indexed	0400	0100 0200 0300	00xx	If disp⩾0, EA = (ptr) If disp<0,EA = (ptr) + disp

xx = —128 to +127

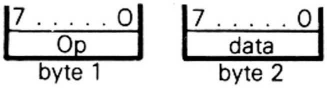
Note: If disp = —128, then (E) is substituted for disp in calculating EA.

The operands for the memory reference instructions are the AC and a memory address.

With these eight instructions the auto-indexed mode of addressing is available; the code is obtained by adding 4 to the code for indexed addressing. If the displacement is positive it is added to the contents of the specified pointer register **after** the contents of the effective address have been fetched or stored. If the displacement is negative it is added to the contents of the pointer register **before** the operation is carried out. This asymmetry makes it possible to implement up to three stacks in memory; values can be pushed onto the stacks or pulled from them with single auto-indexed instructions. Auto-indexed instructions can also be used to add constants to the pointer registers where 16-bit counters are needed.

A special variant of indexed or auto-indexed addressing is provided when the displacement is specified as X'80. In this case it is the contents of the extension register which are added to the specified pointer register to give the effective address. The extension register can thus be used simultaneously as a counter and as an offset to index a table in memory.

For binary addition the 'add' instruction should be preceded by an instruction to clear the CY/L. For binary subtraction the 'complement and add' instruction is used, having first **set** the CY/L. Binary-coded-decimal arithmetic is automatically handled by the 'decimal add' instruction.

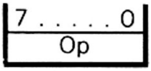


Immediate

Mnemonic	Description	Operation	Op Code Base
LDI	Load Immediate	(AC)←data	C400
ANI	AND Immediate	(AC)←(AC) A data	D400
ORI	OR Immediate	(AC)←(AC) V data	DC00
XRI	Exclusive-OR Immediate	(AC)←(AC) V data	E400
DAI	Decimal Add Immediate	(AC)←$(AC)_{10}$ + $data_{10}$ + (CY/L);(CY/L)	EC00
ADI	Add Immediate	(AC)←(AC) + data + (CY/L);(CY/L),(OV)	F400
CAI	Complement and Add Immediate	(AC)←(AC) + ^data + (CY/L);(CY/L),(OV)	Fc00

Base Code Modifier
Op Code = Base + data

the immediate instructions specify the actual data for the operation in the second byte of the instruction.



Extension Register

Mnemonic	Description	Operation	Op Code
LDE	Load AC from Extension	(AC)←(E)	40
XAE	Exchange AC and Ext.	(AC)↔(E)	01
ANE	AND Extension	(AC)←(AC) A (E)	50
ORE	OR Extension	(AC)←(AC) V (E)	58
XRE	Exclusive-OR Extension	(AC)←(AC) V (E)	60
DAE	Decimal Add Extension	(AC)←$(AC)_{10}$ + $(E)_{10}$ + (CY/L), (CY/L)	68
ADE	Add Extension	(AC)←(AC) + (E) + (CY/L); (CY/L), (OV)	70
CAE	Complement and Add Extension	(AC)←(AC) + ~ (E) + (CY/L); (CY/L), (OV)	78

The extension register can replace the memory address as one operand in the above two-operand instructions. The extension register can be loaded by means of the XAE instruction.

7 . . . 2	1 0
Op	ptr

byte 1

7 0
disp

byte 2

Memory Increment/Decrement

Mnemonic	Description	Operation	Op Code Base
ILD DLD	Increment and Load Decrement and Load	(AC), (EA)←(EA) + 1 (AC), (EA)←(EA) — 1 Note: The processor retains control of the input/output bus between the data read and write operations.	A800 B800

Base Code Modifier

Op Code = Base + ptr + disp

ptr	disp	Effective Address
0100 0200 0300	00xx	EA = (ptr) + disp

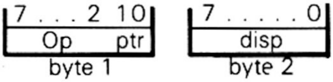
xx = —128 to +127

The 'decrement and load' instruction decrements the contents of the memory location specified by the second byte, leaving the result in the accumulator. This provides a neat way of performing a set of instructions several times. For example:

```
        LDI     9
        ST      COUNT
LOOP:   . . . .
        . . . .
        DLD     COUNT
        JNZ     LOOP
```

will execute the instructions within the loop 9 times before continuing. Both this and the similar 'increment and load' instruction leave the CY/L unchanged so that multibyte arithmetic or shifts can be performed with a single loop.

Transfer



Mnemonic	Description	Operation	Op Code Base
JMP	Jump	(PC)←EA	9000
JP	Jump if Positive	If (AC)≥0, (PC)←EA	9400
JZ	Jump if Zero	If (AC) = 0, (PC)←EA	9800
JNZ	Jump if Not Zero	If (AC) ≠ 0, (PC)←EA	9C00

Base Code Modifier

Op Code = Base + ptr + disp

Address Mode	ptr	disp	Effective Address
PC-relative	0000	00xx	EA = (PC) + disp
Indexed	0100 0200 0300	00xx	EA = (ptr) + disp

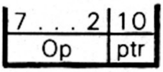
xx = −128 to +127

Transfer of control is provided by the jump instructions which, as with memory addressing, are either PC-relative or relative to one of the pointer registers. Three conditional jumps provide a way of testing the value of the accumulator. 'Jump if positive' gives a jump if the top bit of the AC is zero. The CY/L can be tested with:

```
CSA             ;Copy status to AC
JP      NOCYL   ;CY/L is top of bit status
```

which gives a jump if the CY/L bit is clear.

Pointer Register Move



Mnemonic	Descripton	operation	Op Code Base
XPAL	Exchange Pointer Low	(AC)↔(PTR$_{7:0}$)	30
XPAH	Exchange Pointer High	(AC)↔(PTR$_{15:8}$)	34
XPPC	Exchange Pointer with PC	(PC)↔(PTR)	3C

Base Code Modifier

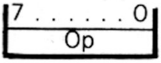
Op Code = Base + ptr

The XPAL and XPAH instructions are used to set up the pointer registers, or to test their contents. For example, to set up P3 to contain X'1234 the following instructions are used:

```
LDI   X'12
XPAH   3
LDI   X'34
XPAL   3
```

The XPPC instruction is used for transfer of control when the point of transfer must be saved, such as in a subroutine call. The instruction exchanges the specified pointer register with the program counter, causing a jump. The value of the program counter is thus saved in the register, and a second XPPC will return control to the calling point. For example, if after the sequence above an XPPC 3 was executed the next instruction executed would be the one at X'1235. Note that this is one beyond the address that was in P3 since the PC is incremented before each instruction. P3 is used by the MK14 monitor to transfer control to the user's program, and an XPPC 3 in the user's program can therefore be used to get back to the monitor provided that P3 has not been altered.

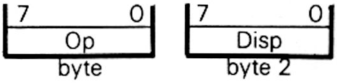
Shift Rotate Serial I/O



Mnemonic	Description	Operation	Op Code
SIO	Serial Input/Output	$(E_i) \rightarrow (E_{i-1})$, SIN → (E_7), (E_0) → SOUT	19
SR	Shift Right	$(AC_i) \rightarrow (AC_{i-1})$, $0 \rightarrow (AC_7)$	1C
SRL	Shift Right with Link	$(AC_i) \rightarrow (AC_{i-1})$, CY/L) → (AC_7)	1D
RR	Rotate Right	$(AC_i) \rightarrow (AC_{i-1})$, $(AC_0) \rightarrow (AC_7)$	1E
RRL	Rotate Right with Link	$(AC_i) \rightarrow (AC_{i-1})$, $(AC_0) \rightarrow$ (CY/L) → (AC_7)	1F

The SIO instruction simultaneously shifts the SIN input into the top bit of the extension register, the bottom bit of the extension register going to the SOUT output; it can therefore form the basis of a simple program to transfer data along a two-way serial line. The shift and rotate with link make possible multibyte shifts or rotates.

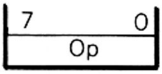
Double Byte Miscellaneous



Mnemonic	Description	Operation	Op Code Base
DLY	Delay	count AC to −1, delay = 13 + 2(AC) + 2 disp + 2^9 disp microcycles	8F00

Base Code Modifier	(= 13 + 2(AC) + 514 DISP µCs.)
Op Code = Base + disp	

The delay instruction gives a delay of from 13 to 131593 microcycles which can be specified in steps of 2 microcycles by the contents of the AC and the second byte of the instruction.
Note that the AC will contain X'FF after the instruction.



Single-Byte Miscellaneous

Mnemonic	Description	Operation	Op Code
HALT	Halt	Pulse H-flag	00
CCL	Clear Carry/Link	(CY/L)←0	02
SCL	Set Carry/Link	(CY/L)←1	03
DINT	Disabled Interrupt	(IE)←0	04
IEN	Enable Interrupt	(IE)←1	05
CSA	Copy Status to AC	(AC)←(SR)	06
CAS	Copy AC to Status	(SR)←(AC)	07
NOP	No Operation	(PC)←(PC) + 1	08

The remaining instructions provide access to the status register, and to the IE and CY/L bits therein. The HALT instruction will act as a NOP in the MK14 kit unless extra logic is added to detect the H-flag at NADS time, in which case it could be used as an extra output. It is used in this way in the single-step circuit; see Section 5.

Mnemonic Index of Instructions

Mnemonic	Opcode	Read Cycles	Write Cycles	Total Microcycles
ADD	F0	3	0	19
ADE	70	1	0	7
ADI	F4	2	0	11
AND	D0	3	0	18
ANE	50	1	0	6
ANI	D4	2	0	10
CAD	F8	3	0	20
CAE	78	1	0	8
CAI	FC	2	0	12
CAS	07	1	0	6
CCI	02	1	0	5
CSA	06	1	0	5
DAD	E8	3	0	23
DAE	68	1	0	11
DAI	EC	2	0	15
DINT	04	1	0	6
DLD	B8	3	1	22
DLY	8F	2	0	13-131593
HALT	00	2	0	8
IEN	05	1	0	6
ILD	A8	3	1	22
JMP	90	2	0	11
JNZ	9C	2	0	9, 11 for Jump
JP	94	2	0	9, 11 for Jump
JZ	98	2	0	9, 11 for Jump
LD	C0	3	0	18
LDE	40	1	0	6
LDI	C4	2	0	10
NOP	08	1	0	5
OR	D8	3	0	18
ORE	58	1	0	6
ORI	DC	2	0	10
RR	1E	1	0	5
RRL	1F	1	0	5
SCL	03	1	0	5
SIO	19	1	0	5
SR	1C	1	0	5
SRL	1D	1	0	5
ST	C8	2	1	18
XAE	01	1	0	7
XOR	E0	3	0	18
XPAH	34	1	0	8
XPAL	30	1	0	8
XPPC	3C	1	0	7
XRE	60	1	0	6
XRI	E4	2	0	10

10 Writing a Program

This section describes the operations involved in designing a program to perform a specific task, and it is hoped that this will provide some guidelines for user wishing to write their own programs. The task chosen is the conversion of numbers from decimal to hexadecimal notation; the result is to be displayed as each digit is entered. For example, to convert 127:

Entered:	Displayed:
'TERM'	0000
'1'	0001 (+1)
'2'	000C (+12)
'7'	007F (+127)

To convert negative numbers to signed twos-complement hexadecimal notation the 'MEM' key will be used as a minus prefix. For example, to convert—127:

Entered:	Displayed:
'MEM'	0000 (—0)
'1'	FFFF (—1)
'2'	FFF4 (—12)
'7'	FF81 (—127)

The procedure is for each new digit entered to multiply the previous total by ten and then add in the value of the new digit. The total will be stored in two memory locations, one for the high-order 8 bits and the other for the low-order 8 bits; with a 16-bit total decimal numbers between +32767 and —32768 can be converted, these numbers being X'7FFF and X'8000 respectively in hexadecimal. For simplicity we will use repeated addition instead of multiplying, adding the previous total to the value of the digit key pressed ten times, and then storing the result back into the running total which will be displayed. A complete flowchart of the program is given in Fig. 10.1.

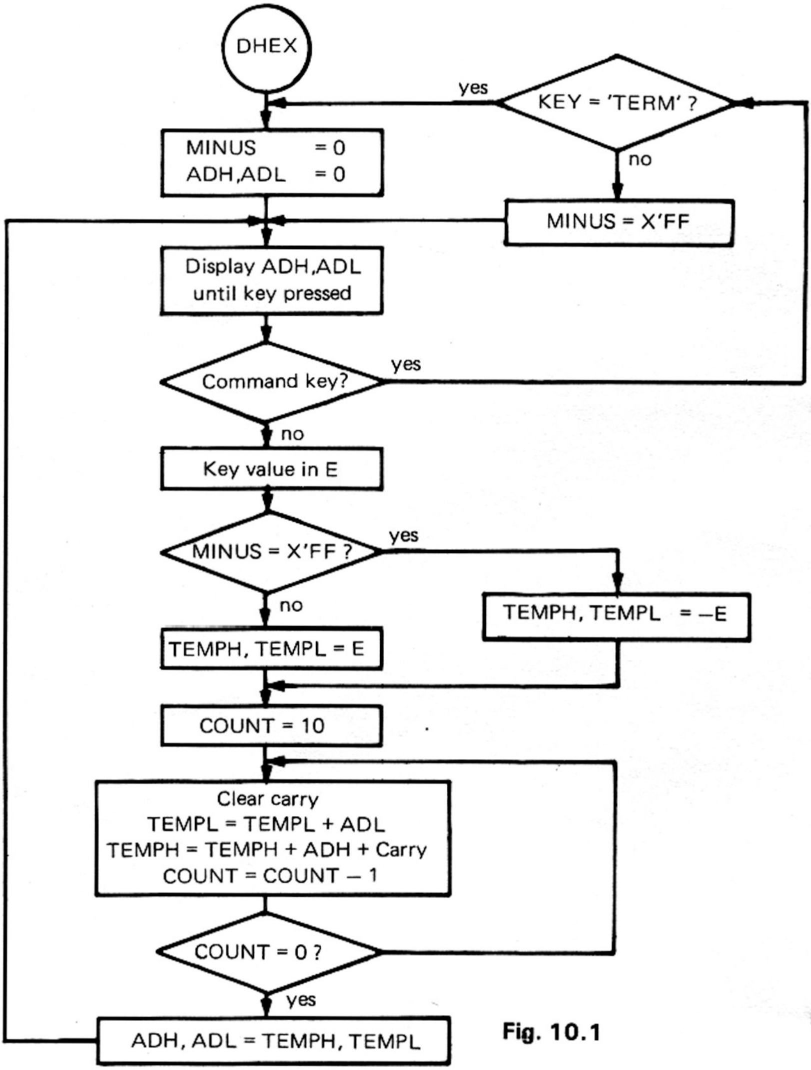


Fig. 10.1

Consider first the part of the program to perform the two-byte addition of the previous total to the new total. Suppose that the high- and low-order bytes of the previous total are referred to as ADH and ADL, and that the corresponding bytes of the new total are TEMPH and TEMPL. Then the program might be:

```
                ; TWO BYTE ADDITION
                ;
0000            ADH         .=0F20
0F20            ADH:        .=.+1
0F21            ADL:        .=.+1
0F22            TEMPH:      .=.+1
0F23            TEMPL:      .=.+1
                ;
0F24   C0FE     ADD2:       LD       TEMPL
0F26   02                   CCL
0F27   F0F9                 ADD      ADL
0F29   C8F9                 ST       TEMPL
0F2B   C0F6                 LD       TEMPH
0F2D   F0F2                 ADD      ADH         ; WITH CARRY
0F2F   C8F2                 ST       TEMPH
0F31   3F                   XPPC     3           ; RETURN TO
                                                 ; MONITOR
```

The addresses of the four memory locations are specified to the load, add, and store instructions using 'program-counter relative' addressing; the second word of the instruction is treated as a displacement to be interpreted as a twos-complement number and added to the value of the program counter to give the effective address for the operation.

Thus the "LD TEMPL" and "ST TEMPL" instructions both address the location TEMPL because X'0F25 + X'FE = X'0F23, and X'0F2A + X'F9 = X'0F23. Only locations within +127 to —128 of the instruction can be addressed using PC-relative addressing, so if the variables needed by a program are to be further away then this it is necessary to use 'indexed addressing'. In this mode of addressing one of the three pointer registers, P1, P2, or P3, is used as a pointer to the variables, and their addresses are specified as displacements from the contents of the pointer register.

To illustrate indexed addressing, assume that P2 has been set up to contain X'0F00 (the start of the RAM); the two-byte addition could then be written:

```
                ; TWO BYTE ADDITION USING INDEXED
                  ADDRESSING
                ; P2 POINTS TO RAM
       0020     ADH         =        X'20
       0021     ADL         =        X'21
       0022     TEMPH       =        X'22
       0023     TEMPL       =        X'23
                ;
0000                        .=0F24
0F24   C223     ADD2:       LD       TEMPL (2)
0F26   02                   CCL
0F27   F221                 ADD      ADL (2)
0F29   CA23                 ST       TEMPL (2)
0F2B   C222                 LD       TEMPH (2)
0F2D   F220                 ADD      ADH (2)     ; WITH CARRY
0F2F   CA22                 ST       TEMPH (2)
0F31   3F                   XPPC     3           ; RETURN TO
                                                 ; MONITOR
```

In this case the four memory locations are specified as offsets from X'0F00 rather than as actual addresses. This is essentially the method used in the final program, except that to save space the extension register is used instead of TEMPH to hold the high-order byte of the new total; this involves changing the "LD TEMPH(2)" and "ST TEMPH(2)" by "LDE" and "XAE" respectively.

It takes four instructions to set up a pointer register with a specific value; for example, to set P2 to X'0F00 the following instructions are needed:

```
         C40F            LDI     X'0F
         36              XPAH    2
         C400            LDI     X'00
         32              XPAL    2
```

Alternatively the monitor program can be used to set the pointer registers as follows. When the monitor is entered by executing an "XPPC 3" instruction at the end of a program it saves the contents of the registers in the top 7 bytes of RAM before using the registers itself.

Similarly, before executing one's own program following the 'GO' command from the keyboard it first loads the registers with the values from these locations. The locations are assigned as follows:

Address:	Stored there:	
0FF9	P1H	High-order byte of P1
0FFA	P1L	Low-order byte of P1
0FFB	P2H	High-order byte of P2
0FFC	P2L	Low order byte of P2
0FFD	A	Accumulator
0FFE	E	Extension register
0FFF	S	Status register

By modifying the contents of these locations before executing a program one can determine what the initial contents of the registers will be. Thus to set up P2 with X'0F00 one stores X'0F at X'0FFB and X'00 at X'0FFC. Pointer P3 is not saved in memory along with the other registers because this contains the return address to the monitor, and if P3 is altered by one's own program it will not be possible to return to the monitor with an "XPPC 3" instruction. Note that pressing 'RESET' zeroes the location X'0FF9—X'0FFF.

Iterative loops

To perform the multiplication by ten one method would be to repeat the instructions for the two-byte addition of ADH, ADL to TEMPH, TEMPL a further nine times. This uses up rather a lot of memory; a better way is to iterate around the same instructions, using a counter to determine when ten iterations have been completed. The program thus becomes:

```
; ADD 10 × ADH,ADL TO TEMPH, TEMPL
; P2 POINTS TO RAM
;
0020  ADH     =          X'20
0021  ADL     =          X'21
0022  TEMPH   =          X'22
0023  TEMPL   =          X'23
```

```
      001F   COUNT     =       X'1F       ; ITERATION
                                          ; COUNTER
             ;
0000                   =0F24
0F24  C40A   MUL10:    LDI     10
0F26  CA1F             ST      COUNT (2)
0F28  C223   ADD2:     LD      TEMPL (2)
0F2A  02               CCL
0F2B  F221             ADD     ADL (2)
0F2D  CA23             ST      TEMPL (2)
0F2F  C222             LD      TEMPH (2)
0F31  F220             ADD     ADH (2)    ; WITH CARRY
0F33  CA22             ST      TEMPH (2)
0F35  BA1F             DLD     COUNT (2)
0F37  9CEF             JNZ     ADD 2      ; MORE ITERATIONS
                                            TO DO
0F39  3F               XPPC    3          ; ELSE RETURN
```

Again P2 is assumed to contain X'0F00 before this is executed. Now all we have still to do is to start by loading TEMPH,TEMPL with the value of the key pressed, and afterwards to store TEMPH,TEMPL back into ADH,ADL and display this result as four hexadecimal digits.

The Display Interface

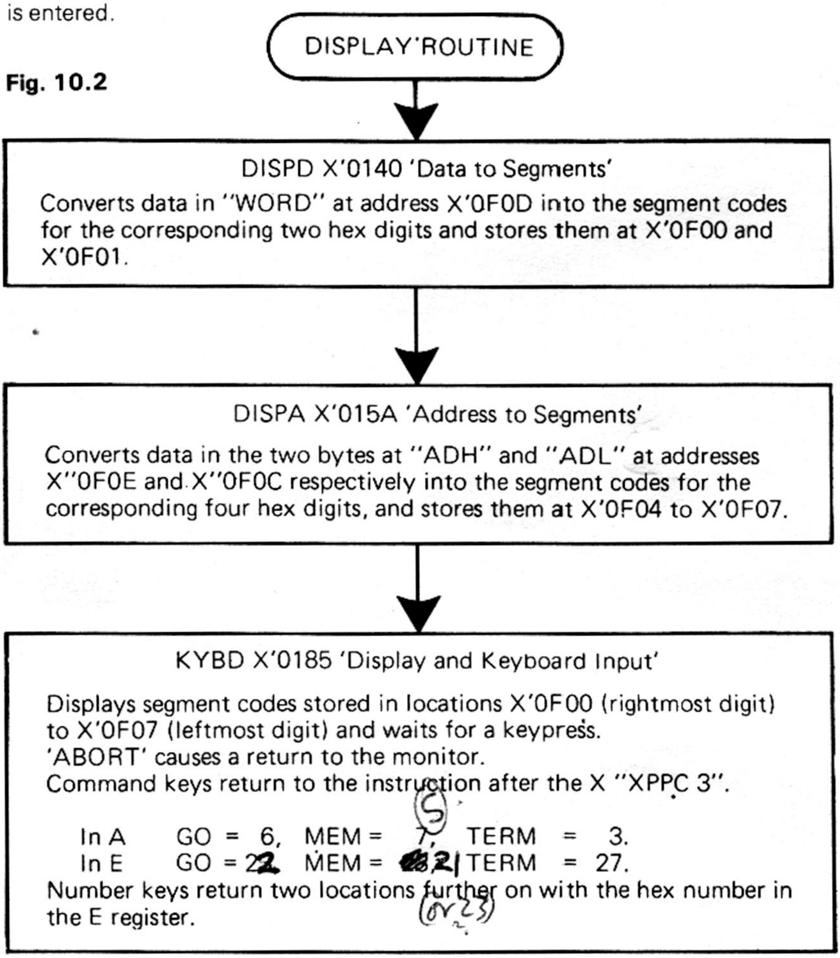
The keyboard and display are addressed by the microprocessor just like a row of eight consecutive memory locations, X'0D00 to X'0D07; X'0D00 controls the rightmost digit and X'0D07 the leftmost digit. To illuminate a digit a binary code is stored at the address corresponding to that digit; each of the lower seven bits controls one of the segments of the display digit, the lowest bit controlling the 'a' segment up to bit 6 controlling the 'g' segment. Thus any combination of the segments of any digit may be illuminated, making it possible to generate some of the letters of the alphabet as well as the hex digits. Only one display digit is lit up at any one time, so to generate the appearance of a static display of eight digits the eight display addresses must be repeatedly written to with the required eight segment codes. The following simple program demonstrates how the display may be driven directly; pointer P1 is set up to point to the display.

```
             ; DISPLAY DEMONSTRATION
             ;P1 POINTS  TO DISPLAY
             ;
0000                   =0F20
0F20  8F80   DEMO;     DLY     x'80       ; DETERMINES
                                          ; SPEED
0F22  A802             ILD     STORE+1
0F24  C900   STORE:    ST      X'00 (1)
0F26  90F8             JMP     DEMO       ; LOOP FOREVER
0F28                   .=0FF9
0FF9  0D00             .DBYTE  X'0D00     ; SO MONITOR
                                          ; SETS P1
```

This program changes the displacement at X'0F25 to select a different display digit and to determine the character generated. The applications programs which generate a changing display write directly to the display addresses in a similar way; see for example: 'Duck Shoot', p. 72 'Digital

Alarm Clock', p. 55, and 'Message', p. 84.
To display the total in the decimal-to-hex program we would first need to generate the four segment codes corresponding to the four hexadecimal digits to be displayed, and then write them repeatedly to the display addresses to give the display. Fortunately there is a routine in the monitor which will perform this task, and as a bonus it will wait until a key is pressed and then return with its value. The flowchart in Fig. 10.2 indicates what the routine does and this depends on the stage at which it is entered.

Fig. 10.2



DISPLAY'ROUTINE

DISPD X'0140 'Data to Segments'

Converts data in "WORD" at address X'0F0D into the segment codes for the corresponding two hex digits and stores them at X'0F00 and X'0F01.

DISPA X'015A 'Address to Segments'

Converts data in the two bytes at "ADH" and "ADL" at addresses X"0F0E and X"0F0C respectively into the segment codes for the corresponding four hex digits, and stores them at X'0F04 to X'0F07.

KYBD X'0185 'Display and Keyboard Input'

Displays segment codes stored in locations X'0F00 (rightmost digit) to X'0F07 (leftmost digit) and waits for a keypress.
'ABORT' causes a return to the monitor.
Command keys return to the instruction after the X "XPPC 3".

(5)
In A GO = 6, MEM = 7, TERM = 3.
In E GO = 22 MEM = 21 TERM = 27.
Number keys return two locations further on with the hex number in the E register.
(or 23)

The RAM addresses given in the flowchart assume that P2 has been set up to contain X'0F00. To call the routine P3 is first set up with the required entry address minus one, and then an "XPPC 3" instruction is executed which exchanges this value with the value in the program counter. The reason for loading one less than the address is that the PC is incremented before the execution of every instruction. As an example, the following program will cause '0123 45' to be displayed:

```
                    ; DISPLAY HEX NUMBERS USING DISPD
                    ; P2 SET TO POINT TO RAM
                    ;
             000E   ADH         =         X'0E
```

```
          000C  ADL          =      X'0C
          000D  WORD         =      X'0D
                ;
                             =0F20
0F20      C401  DISHEX:      LDI    X'01
0F22      CA0E               ST     ADH (2)
0F24      C423               LDI    X'23
0F26      CA0C               ST     ADL (2)
0F28      C445               LDI    X'45
0F2A      CA0D  SHOW:        ST     WORD (2)
0F2C      C401               LDI    X'01       ; H(DISPD)
0F2E      37                 XPAH   3
0F2F      C43F               LDI    X'3F       ; L(DISPD)—1
0F31      33                 XPAL   3
0F32      3F                 XPPC   3          ; JUMP TO DISPD
                                               ; SUBROUTINE
0F33      90F5               JMP    SHOW       ; COMMAND KEY
                                               ; RETURN
0F35      01                 XAE               ; NUMBER KEY
                                               ; RETURN
0F36      90F2               JMP    SHOW
                ;
0F38                         =0FFB
0FFB      0F00               .DBYTE X'0F00     ; SO MONITOR
                                               ; SETS P2
```

When a key is pressed the point of return distinguishes whether it was a command key or a number key. Command keys (except 'ABORT') return to the instruction after the ''XPPC 3''. Numerical keys return two bytes after that address, leaving room to put in a ''JMP'' to the part of the program that deals with command keys. In the above program ''DISHEX'' the value in the accumulator for command keys, and the value in the E register for number keys, is stored at ''WORD'' so that it will be displayed in the rightmost two digit positions; so, for example, pressing 'MEM' will change the display to '0123 07''.

In the decimal-to-hex program we need to display a two-byte number as four hex digits, so the ''DISPA'' entry point to the display routine will be used. The only operation remaining is to set a flag if the 'MEM' key has been pressed, indicating that a negative decimal number is to be entered, and if the flag is set to negate the value of each numerical key pressed before repeatedly adding it to the running total. It is convenient to make the value of this ''MINUS'' flag X'00 for positive numbers and X'FF for negative ones. Then if the key's value is in the accumulator and the value of the ''MINUS'' flag is in the E register, the sequence: ''SCL, XRE, CAE'' will negate it if ''MINUS'' is X'FF and leave it unchanged if ''MINUS'' is zero. The high-order byte of the key's value must be X'00 if the number is positive and X'FF if it is negative, and at first sight it looks as if the value of ''MINUS'' would do; but then entering 'MEM', '0' would set the key's value to X'FF00 which is incorrect (it should be X'0000). Instead we make use of the carry from the previous operation, and ''LDI 0, CAD 0'' sets the accumulator to the correct value for the key's high-order byte. Since the operand for these two instructions could be anything, not necessarily zero, the sequence ''LDE, CAE'' would do just as well, irrespective of what the E register contains, and this is used in the final program to save a couple of bytes.

The program is now virtually complete, and the full listing given on page 78 of the applications manual should be recognizable as an amalgamation of the sections that have been discussed above. The program can in fact be used to convert numbers from any base into hexadecimal by changing the multiplication factor at X'0F62 from its present value of X'0A (ten) to the value of the base from which conversion is required.

11 RAM I/O

A socket is provided on the MK14 to accept the 40 pin RAM I/O device (manufacturers part no. INS8154). This device can be added without any additional modification, and provides the kit user with a further 128 words of RAM and a set of 16 lines which can be utilised as logic inputs in any combination.

These 16 lines are designated Port A (8 lines) and Port B (8 lines) and are available at the edge connector as shown in Fig. 11.1.

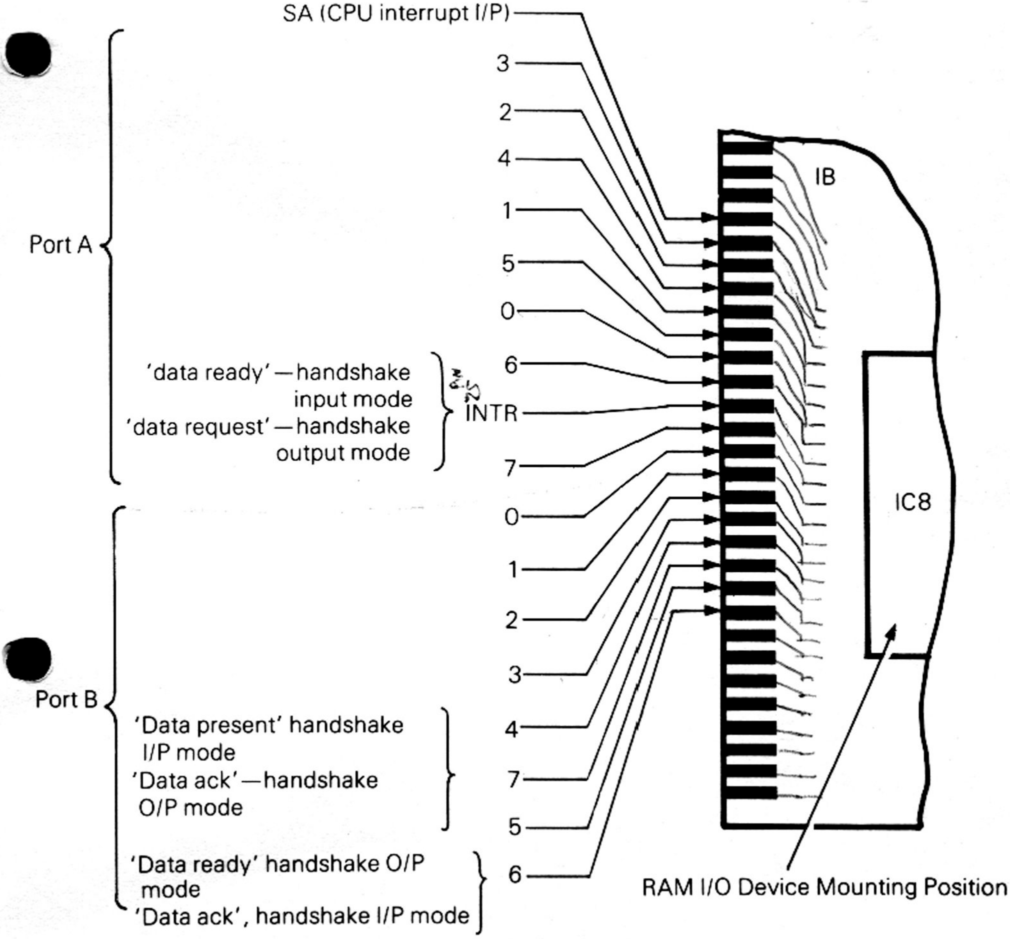
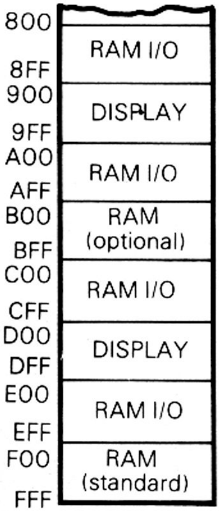

SA (CPU interrupt I/P)
Port A
3
2
4
1
5
0
'data ready'—handshake input mode
'data request'—handshake output mode
6
INTR
7
Port B
0
1
2
3
'Data present' handshake I/P mode
'Data ack'—handshake O/P mode
4
7
5
'Data ready' handshake O/P mode
'Data ack', handshake I/P mode
6
IB
IC8
RAM I/O Device Mounting Position

Fig. 11.1 RAM I/O Signal Lines

The RAM I/O can be regarded as two completely separate functional entities, one being the memory element and the other the input/output section. The only association between the two is that they share the same package and occupy adjacent areas in the memory I/O space. Fig. 11.2 shows the blocks in the memory map occupied by the RAM I/O, and it can be seen that the one piece of hardware is present in four separate blocks of memory.

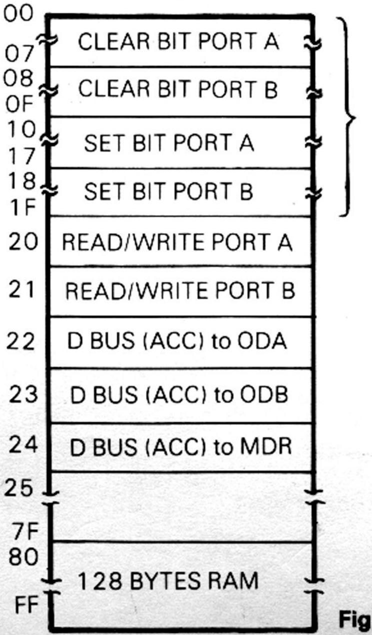


Note:—Memory area is shown divided into 256 byte blocks. The lowest and highest location address is shown in hex' at left.

Fig. 11.2 Memory I/O Map Showing RAM I/O Areas

The primary advantage for the user, in this, is that program located in basic RAM, or in the extra RAM option, has the same address relationship to the RAM I/O.

Fig. 11.3 shows how memory I/O space within the RAM I/O block is allocated.



Selected bit out of 8 determined by low 3 bits of address
e.g. Addr. = 0, bit = 0 (Port A)
Addr. = IF, bit = 7 (Port B)

Fig. 11.3 RAM I/O Locations and Related Functions

RAM Section

This is utilised in precisely the same manner as any other area of RAM.

Input/Output Section

The device incorporates circuitry which affords the user a great deal of flexibility in usage of the 1 6 input/output lines. Each line can be separately defined as either an input or an output under program control. Each line can be independently either read as an input, or set to logic 'I' or 'O' as an output. These functions are determined by the address value employed.

A further group of usage modes permit handshake logic i.e. a 'data request', 'data ready', 'data receieved', signalling sequence to take place in conjunction with 8 bit parallel data transfers in or out through Port A.

Reset Control

This input from the RAM I/O is connected in parallel with the CPU power-on and manual reset. When reset is present all port lines are high impedance and the device is inhibited from all operations.

Following reset all port lines are set to input mode, handshake facilities are deselected and all port output latches are set to zero.

Input/Output Definition Control

At start-up all 1 6 lines will be in input mode. To convert a line or lines to the output condition a write operation must be performed by program into the ODA (output definition port A) or ODB locations e.g. writing the value 80 (Hex.) into ODB will cause bit 7 port B to become an output.

Single Bit Read

The logic value at an input pin is transferred to the high order bit (bit 7) by performing a read instruction. The remaining bits in the accumulator become zero.

The required bit is selected by addressing the appropriate location (see Figs. 3 & 4).

By executing JP (Jump if Positive) instruction the program can respond to the input signal i.e. the jump does not occur if the I/P is a logic '1'

If a bit designated as an output is read the current value of that O/P is detected.

Single Bit Load

This is achieved by addressing a write operation to a selected location (see Figs. 11.1 & 11.4). Note that it is not necessary to preset the accumulator to define the written bit value because it is determined by bit 4 of the address.

Eight Bit Parallel Read or Write

An eight bit value can be read from Port A or B to the accumulator, or the accumulator value can be output to Port A or B. See Figs. 11.3 & 11.4 for the appropriate address locations. Input/output lines must be pre-defined for the required mode.

Port A Handshake Operations

To achieve eight bit data transfers with accompanying handshake via Port A, two lines (6 and 7) from Port B are allocate special functions and must be pre-defined by program as follows:- bit 7-input, bit 6-output.

Additionally the INTR signal line is utilised.

Three modes of handshake function are available to be selected under program control. Fig. 11.4 shows values to be written into the three higher order bits of the Mode Definition Register (see Fig. 11.1 for location) for the various modes.

Bit Position & value in MDR

	7	6	5
BASIC I/O this condition selected by reset	X	X	O
STROBED INPUT	X	O	I
STROBED OUTPUT	O	I	I
STROBED OUTPUT WITH TRI-STATE	I	I	I

Note:-
i) X = don't care
ii) Lower order bits are don't care also.

Fig. 11.4 Mode Definition Register (MDR) Values and Operation Modes

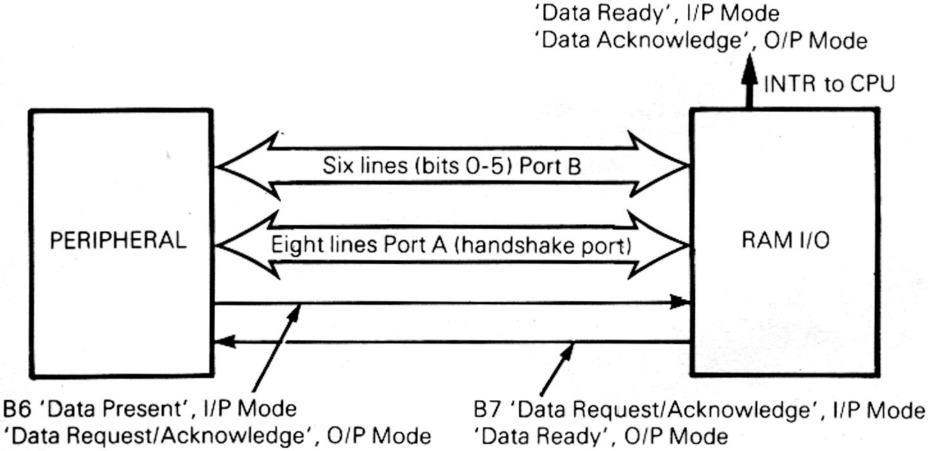

'Data Ready', I/P Mode
'Data Acknowledge', O/P Mode
INTR to CPU
PERIPHERAL
Six lines (bits 0-5) Port B
Eight lines Port A (handshake port)
RAM I/O
B6 'Data Present', I/P Mode
'Data Request/Acknowledge', O/P Mode
B7 'Data Request/Acknowledge', I/P Mode
'Data Ready', O/P Mode

Fig. 11.5 Handshake Interconnections and Function

INTR Signal

In order to inform the CPU of the state of the data transfer in handshake mode the RAM I/O generates the INTR SIGNAL: This signal will usually be connected to the CPU interrupt input SA.

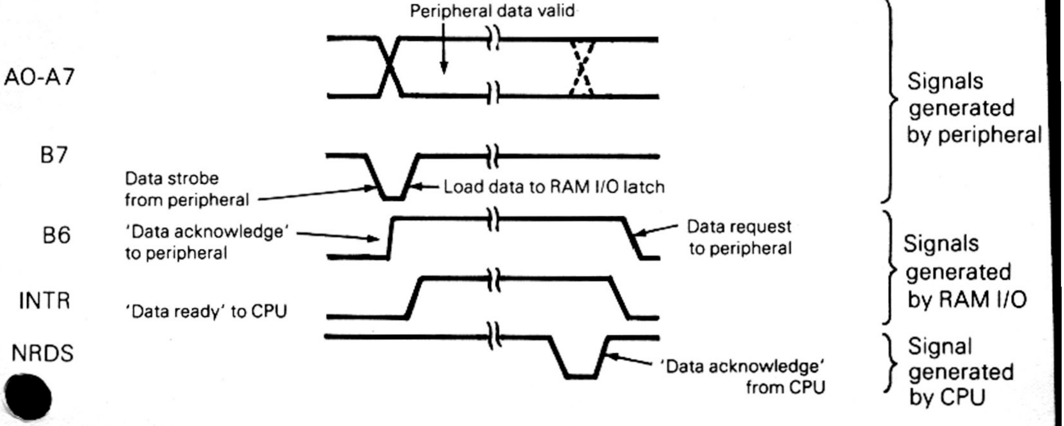
The INTR signal is activated by writing a logic 'I' into B7 and is inhibited by a logic 'O'. Note that although B7 must be defined as an input, in handshake mode the B7 output latch remains available to perform this special function.

Strobed Input Mode

A peripheral circuit applies a byte of information to Port A and a low pulse to B7. The pulse causes the data to be latched into the RAM I/O Port A register, and B6 is made high as a signal to the peripheral indicating that the latch is now occupied. At the same time INTR (if enabled) goes high indicating 'data ready' to the CPU.

The CPU responds with a byte read from Port A. The RAM I/O recognises this, and removes INTR and the 'buffer full' signal on B6, informing the peripheral that the latch is available for new data.

Fig. 11.6 Signal Timing Relationship—Handshake I/P Mode


Peripheral data valid
AO-A7
B7
Data strobe from peripheral
Load data to RAM I/O latch
B6
'Data acknowledge' to peripheral
Data request to peripheral
INTR
'Data ready' to CPU
NRDS
'Data acknowledge' from CPU
Signals generated by peripheral
Signals generated by RAM I/O
Signal generated by CPU

Strobed Output Mode

The CPU performs a byte write to Port A, and the RAM I/O generates a 'data ready' signal by making B6 low. The peripheral responds to 'data ready' by accepting the Port A data, and acknowledges by making B7 low. When B7 goes low the RAM I/O makes INTR high (if enabled) informing the CPU that the data transaction is complete.

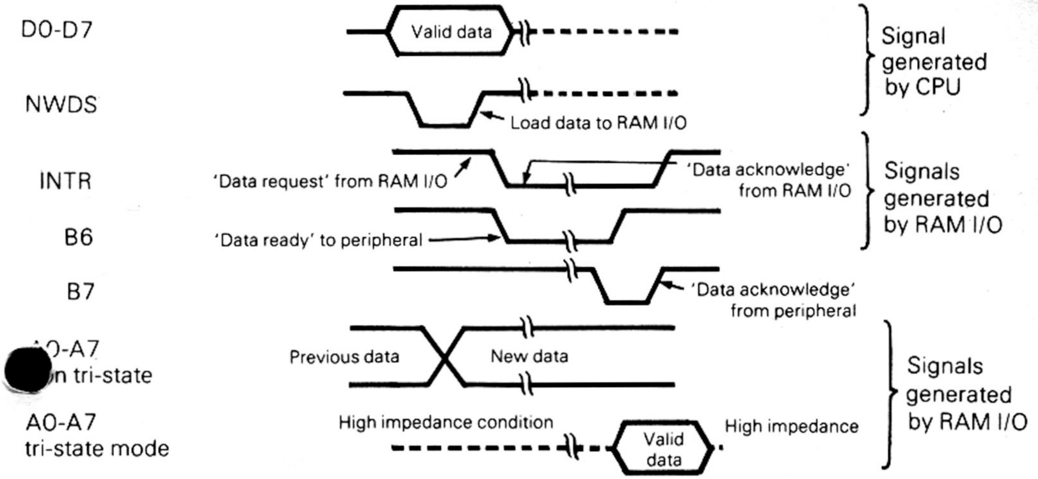

DO-D7
Valid data
NWDS
Load data to RAM I/O
INTR
'Data request' from RAM I/O
'Data acknowledge' from RAM I/O
B6
'Data ready' to peripheral
B7
'Data acknowledge' from peripheral
0-A7
n tri-state
Previous data
New data
AO-A7
tri-state mode
High impedance condition
Valid data
High impedance
Signal generated by CPU
Signals generated by RAM I/O
Signals generated by RAM I/O

Fig. 11.7Signal Timing Relationship—Handshake O/P Mode

Strobed Output with Tri-State Control

This mode employs the same signalling and data sequence as does Output Mode above. However the difference lies in that Port A will, in this mode, normally be in Tri-state condition (i.e. no load on peripheral bus), and will only apply data to the bus when demanded by the peripheral by a low acknowledge signal to B7.

Applications for Handshake Mode

Handshake facilities afford the greatest advantages when the MK14 is interfaced to an external system which is independent to a greater or lesser degree. Another MK14 would be an example of an completely independent system.

In comparison the simple read or write, bit or byte, modes are useful when the inputs and outputs are direct connections with elements that are subservient to the MK14.

However whenever the external system is independently generating and processing data the basic 'data request', 'data ready', 'data acknowledge', sequence becomes valuable. The RAM I/O first of all relieves the MK14 software of the task of creating the handshake. Secondly it is likely in this kind of situation that the MK14 and external system are operating asynchronously i.e. are not synchronised to a common time source or system protocol. This implies that when one element is ready for a data transfer, the other may be busy with some other task.

Here the buffering ability of the Port A latch eases these time constraints by holding data transmitted by one element until the other is ready to receive.

Therefore, for example, if the CPU, in the position of a receiver, is unable, due to the requirements of the controlling software, in the worst case, to pay attention for 2 millisecs the transmitter would be allowed to send data once every millisecond.

Part 2

Devised and written by:
David Johnson—Davies
except programs marked thus*

Monitor program listing

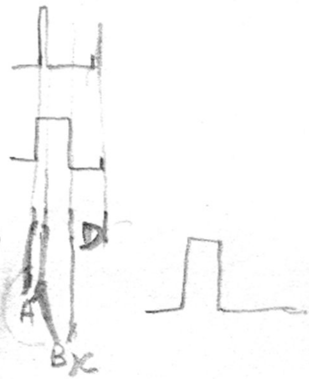
SCIOS

```
                TITLE SCIOS
;*********************************************************
;*
;*
;*     DEVELOPED FROM SCMPKB MONITOR
;*     BY D.J.D.
;*     TAPE ROUTINES BY N.J.T.
;*
;*
;*********************************************************


   0F00  RAM      =     0F00
   0D00  DISP     =     0D00
        ;         RAM OFF-SET
   0000  DL       =     0              ;SEGMENT FOR DIGIT 1
   0001  DH       =     1              ;SEGMENT FOR DIGIT 2
   0002  D3       =     2              ;SEGMENT FOR DIGIT 3
   0003  D4       =     3              ;SEGMENT FOR DIGIT 4
   0004  ADLL     =     4              ;SEGMENT FOR DIGIT 5
   0005  ADLH     =     5              ;SEGMENT FOR DIGIT 6
   0006  ADHL     =     6              ;SEGMENT FOR DIGIT 7
   0007  ADHH     =     7              ;SEGMENT FOR DIGIT 8
   0008  D9       =     8              ;SEGMENT FOR DIGIT 9
   0009  CNT      =     9              ;COUNTER.
   000A  PUSHED   =     10             ;KEY PUSHED.
   000B  CHAR     =     11             ;CHAR READ.
   000C  ADL      =     12             ;MEMORY ADDRESS LOW.
   000D  WORD     =     13             ;MEMORY WORD.
   000E  ADH      =     14             ;MEMORY ADDRESS HI.
   000F  DDTA     =     15             ;FIRST FLAG.
   0010  ROW      =     16             ;ROW COUNTER.
   0011  NEXT     =     17             ;FLAG FOR NOW DATA.
         RAM POINTERS USED BY SCIOS, P3 IS SAVED ELSEWHERE
   0FF9  P1H      =    0FF9
   0FFA  P1L      =    0FFA
   0FFB  P2H      =    0FFB
   0FFC  P2L      =    0FFC
   0FFD  A        =    0FFD
   0FFE  E        =    0FFE
   0FFF  S        =    0FFF
```

```
;        MONITOR OPERATION SUMMARY
;
;        INITIALLY IN 'ADDRESS ENTRY' MODE
;
;TERM
;        CHANGE TO 'DATA-ENTRY' MODE
;
;MEM:
;        INCREMENT MEMORY ADDRESS
;
;ABORT:
;        CHANGE TO 'ADDRESS ENTRY' MODE
;
;GO:
;
;            THE REGISTERS ARE LOADED FROM RAM AND PROGRAM
;            IS TRANSFERRED USING XPPC P3.
;            TO GET BACK DO A XPPC P3.

;            MONITOR LISTING

0000  00              HALT              ;ZEROS DISPLAYED ON RESET

0001  CFFF INIT:      ST @—1(3)   ;SO P3 = —1


0003 901E             JMP START

0005
                ;     DEBUG EXIT
                ;     RESTORE ENVIRONMENT
            GOOUT:
0005  37              XPAH  3
0006  C20C            LD    ADL(2)
0008  33              XPAL  3
0009  C7FF            LD    @-1(3)     ;FIX GO ADDRESS.
000B  COF2            LD    E          ;RESTORE REGISTERS.
000D  01              XAE
000E  COEB            LD    P1L
0010  31              XPAL  1
0011  COE7            LD    P1H
0013  35              XPAH  1
0014  COE7            LD    P2L
0016  32              XPAL  2
0017  COE3            LD    P2H
0019  36              XPAH  2
001A  COE4            LD    S
```

```
001C 00               HALT              ;RESET SINGLE-STEP
001D 07               CAS
001E CODE             LD     A
0020 08               NOP
0021 05               IEN
0022 3F               XPPC   3
                ;     ENTRY POINT FOR DEBUG
0023            START:
0023 C8D9             ST     A
0025 40               LDE
0026 C8D7             ST     E
0028 06               CSA
0029 C8D5             ST     S
002B 35               XPAH   1
002C C8CC             ST     P1H
002E 31               XPAL   1
002F C8CA             ST     P1L
0031 C40F             LDI    H(RAM)     ;POINT P2 TO RAM
0033 36               XPAH   2
0034 C8C6             ST     P2H
0036 C400             LDI    L(RAM)
0038 32               XPAL   2
0039 C8C2             ST     P2L
003B C701             LD     @1(3)      ;BUMP P3 FOR RETURN
003D 33               XPAL   3          ;SAVE P3
003E CAOC             ST     ADL(2)
0040 37               XPAH   3
0041 CAOE             ST     ADH(2)
0043 C400             LDI    0
0045 CA02             ST     D3(2)
0047 CA03             ST     D4(2)
0049 C401             LDI    1
004B 37               XPAH   3
004C            ABORT:
004C 906D             JMP    MEM
004E            GONOW:
004E C20E             LD     ADH(2)
0050 90B3             JMP    GOOUT
001C 00

                      ;
                      ;      TAPE INTERFACE ROUTINES
                      ;
     00D5             COUNT  =      0D5  0FF7
     00D6             LEN    =      0D6  0FF8
                      ;
                      ;      STORE TO TAPE = 0052
                      ;
0052 C501             TOTAPE: LD1(1)
0054 01                      XAE
0055 C401                    LDI 1
0057 CBD5             NEXT:  ST COUNT(3)
0059 C401                    LDI 1
```

```
                          (x4)  (x8)
005B  07                                   CAS
005C  8F08              A (02)  (01)       DLY 8
005E  C3D5                                 LD COUNT(3)
0060  50                                   ANE
0061  9807                                 JZ ZERO
0063  8F18              B (06)  (03)       DLY 018
0065  C400                                 LDI 0
0067  07                                   CAS
0068  9005                                 JMP DONE
006A  C400                      ZERO:      LDI 0
006C  07                                   CAS
006D  8F18             C (06)  (03)        DLY 018
006F  8F20             D (08)   DONE:      DLY 020
0071  C3D5                      (04)       LD COUNT(3)
0073  F3D5                                 ADD COUNT(3)
0075  9CE0                                 JNZ NEXT
0077  BBD6                                 DLD LEN(3)
0079  9CD7                                 JNZ TOTAPE
007B  3F                                   XPPC 3
                                ;
                                ;          LOAD FROM TAPE = 007C
                                ;
007C  C408                      FRTAPE:    LDI 8
007E  CBD5                                 ST COUNT(3)
0080  06                        LOOP:      CSA
0081  D420              (x4)    (x8)       ANI 20
0083  98FB                                 JZ LOOP
0085  8F1C              0C      07         DLY 01C
0087  19                                   S10
0088  8F1C                                 DLY 01C
008A  BBD5                                 DLD COUNT(3)
008C  9CF2                                 JNZ LOOP
008E  40                                   LDE
008F  CD01                                 ST@1 (1)
0091  90E9                                 JMP FRTAPE
                    ;
                    ;          OFFSET  CALCULATION = 0093
                    ;
0093                OFFSET:
0093  C6FE                     LD@-2 (2)       ;Subtract 2 from
                                               ;destination address
0095  32                       XPAL 2          ;Put low byte in AC
0096  03                       SCL             ;Set carry for subtraction
0097  FBD8                     CAD OD8(3)      ;Subtract low byte of jump
                                               ;instruction address
0099  C901                     ST + 1 (1)      ;Put in jump operand
009B  3F                       XPPC 3          ;Return to monitor
                                  ;
009C  08                       NOP
                                  ;
009D                DTACK:
009D  AAOE                     ILD   ADH(2)
009F  9036                     JMP   DATA
```



```
00A1           MEMDN:
00A1  C20E               LD     ADH(2)      ;PUT WORD IN MEM.
00A3  35                 XPAH   1
00A4  C20C               LD     ADL(2)
00A6  31                 XPAL   1
00A7  C20D               LD     WORD(2)
00A9  C900               ST     (1)
00AB  9034               JMP    DATACK

00AD           MEMCK:
00AD  E406               XRI    06          ;CHECK FOR GO.
00AF  989D               JZ                 ; GONOW
00B1  E405               XRI    05          ;CHECK FOR TERM.
00B3  9822               JZ     DATA        ;CHECK IF DONE.
00B5  AA0C               ILD    ADL(2)      ;UPDATE ADDRESS LOW.
00B7  9C1E               JNZ    DATA
00B9  90E2               JMP    DTACK
               ;         MEM KEY PUSHED
00BB           MEM:
00BB  C4FF               LDI    -1          ;SET FIRST FLAG.
00BD  CA11               ST     NEXT(2)     ;SET FLAG FOR ADDRESS NOW.
00BF  CA0F               ST     DDTA(2)
00C1           MEML:
00C1  C20E               LD     ADH(2)
00C3  35                 XPAH   1           ;SET P1  FOR MEM ADDRESS.
00C4  C20C               LD     ADL(2)
00C6  31                 XPAL   1
00C7  C100               LD     (1)
00C9  CA0D               ST     WORD(2)     ;SAVE MEM DATA.
00CB  C43F               LDI    L(DISPD)-1  ;FIX DATA SEG.
00CD  33                 XPAL   3
00CE  3F                 XPPC   3           ;GO TO DISPD SET SEG FOR DATA.
00CF  90DC               JMP    MEMCK       ;COMMAND RETURN.
00D1  C41A               LDI    L(ADR)-1    ;MAKE ADDRESS.
00D3  33                 XPAL   3
00D4  3F                 XPPC   3
00D5  90EA               JMP    MEML        ;GET NEXT CHAR.
00D7           DATA:
00D7  C4FF               LDI    -1          ;SET FIRST FLAG.
00D9  CA0F               ST     DDTA(2)
00DB  C20E               LD     ADH(2)      ;SET P1  TO MEMORY ADDRESS
00DD  35                 XPAH   1
00DE  C20C               LD     ADL(2)
00E0  31                 XPAL   1
00E1  C100   ←DATACK:    LD     (1)         ;READ DATA WORD.
00E3  CA0D               ST     WORD(2)     ;SAVE FOR DISPLAY.

                         .PAGE
00E5           DATAL:
00E5  C43F               LDI    L(DISPD)-1  ;FIX DATA SEG.
00E7  33                 XPAL   3
00E8  3F                 XPPC   3           ;FIX DATA SEG-GO TO DISPD.
```

```
00E9  90C2                JMP    MEMCK       ;CHAR RETURN.
00EB  C404                LDI    4           ;SET COUNTER FOR NUMBER OF SHIFTS.
00ED  CA09                ST     CNT(2)
00EF  AA0F                ILD    DDTA(2)     ;CHECK IF FIRST.
00F1  9C06                JNZ    DNFST
00F3  C400                LDI    0           ;ZERO WORD IF FIRST.
00F5  CA0D                ST     WORD(2)
00F7  CA11                ST     NEXT(2)     ;SET FLAG FOR ADDRESS DONE.
00F9        DNFST:
00F9  02                  CCL
00FA  C20D                LD     WORD(2)     ;SHIFT LEFT.
00FC  F20D                ADD    WORD(2)
00FE  CA0D                ST     WORD(2)
0100  BA09                DLD    CNT(2)      ;CHECK FOR 4 SHIFTS.
0102  9CF5                JNZ    DNFST
0104  C20D                LD     WORD(2)     ;ADD NEW DATA.
0106  58                  ORE
0107  CA0D                ST     WORD(2)
0109  9096                JMP    /MEMDN
            ;             SEGMENT ASSIGNMENTS
      0001  SA             =       1
      0002  SB             =       2
      0004  SC             =       4
      0008  SD             =       8
      0010  SE             =       16
      0020  SF             =       32
      0040  SG             =       64

            ;             'HEX NUMBER TO  SEVEN SEGMENT TABLE'

      010B          CROM:
      010B  3F (Ø)               .BYTE     SA+SB+SC+SD+SE+SF
      010C  06 (1)               .BYTE     SB+SC
      010D  5B (2)               .BYTE     SA+SB+SD+SE+SG
      010E  4F (3)               .BYTE     SA+SB+SC+SD+SG
      010F  66 (4)               .BYTE     SB+SC+SF+SG
      0110  6D (5)               .BYTE     SA+SC+SD+SF+SG
      0111  7D (6)               .BYTE     SA+SC+SD+SE+SF+SG
      0112  07 (7)               .BYTE     SA+SB+SC
      0113  7F (8)               .BYTE     SA+SB+SC+SD+SE+SF+SG
      0114  67 (9)               .BYTE     SA+SB+SC+SF+SG
      0115  77 (A)               .BYTE     SA+SB+SC+SE+SF+SG
      0116  7C (B)               .BYTE     SC+SD+SE+SF+SG
      0117  39 (C)               .BYTE     SA+SD+SE+SF
      0118  5E (D)               .BYTE     SB+SC+SD+SE+SG
      0119  79 (E)               .BYTE     SA+SD+SE+SF+SG
      011A  71 (F)               .BYTE     SA+SE+SF+SG

                                 .PAGE  'MAKE 4 DIGIT ADDRESS'
      011B          ADR:
                    ;            SHIFT ADDRESS LEFT ONE DIGIT THEN
                    ;            ADD NEW LOW HEX DIGIT.
                    ;            HEX DIGIT IN E REGISTER.
                    ;            P2 POINTS TO RAM.
```

```
011B C404              LDI    4          ;SET NUMBER OF SHIFTS.
011D CA09              ST     CNT(2)
011F AA0F              ILD    DDTA(2)    ;CHECK IF FIRST.
0121 9C06              JNZ    NOTFST     ;JMP IF NO.
0123 C400              LDI    0          ;ZERO ADDRESS.
0125 CA0E              ST     ADH(2)
0127 CA0C              ST     ADL(2)
0129         NOTFST:
0129 02                CCL               ;CLEAR LINK.
012A C20C              LD     ADL(2)     ;SHIFT ADDRESS LEFT 4 TIMES.
012C F20C              ADD    ADL(2)
012E CA0C              ST     ADL(2)     ;SAVE IT.
0130 C20E              LD     ADH(2)     ;NOW SHIFT HIGH.
0132 F20E              ADD    ADH(2)
0134 CA0E              ST     ADH(2)
0136 BA09              DLD    CNT(2)     ;CHECK IF SHIFTED 4 TIMES.
0138 9CEF              JNZ    NOTFST     ;JMP IF NOT DONE.
013A C20C              LD     ADL(2)     ;NOW ADD NEW NUMBER.
013C 58                ORE
013D CA0C              ST     ADL(2)     ;NUMBER IS NOW UP DATED.
013F 3F                XPPC   3
                       .PAGE  'DATA TO SEGMENTS'


             ;         CONVERT HEX DATA TO SEGMENTS.
             ;         P2 POINTS TO RAM.
             ;         DROPS THRU TO HEX ADDRESS CONVERSION.


0140         DISPD:
0140 C401              LDI    H(CROM)    ;SET ADDRESS OF TABLE.
0142 35                XPAH   1
0143 C40B              LDI    L(CROM)
0145 31                XPAL   1
0146 C20D              LD   WORD(2)      ;GET MEMORY WORD.
0148 D40F              ANI    OF
014A 01                XAE
014B C180              LD     -128(1)    ;GET SEGMENT DISP.
014D CA00              ST     DL(2)      ;SAVE AT DATA LOW.
014F C20D              LD     WORD(2)    ;FIX HI.
0151 1C                SR                ;SHIFT HI TO LOW.
0152 1C                SR
0153 1C                SR
0154 1C                SR
0155 01                XAE
0156 C180              LD     -128(1)    ;GET SEGMENTS.
0158 CA01              ST     DH(2)      ;SAVE IN DATA HI.
             .PAGE     ADDRESS TO SEGMENTS
```

```
                      ;           CONVERT HEX ADDRESS TO SEGMENTS.
                      ;           P2 POINTS TO RAM.
                      ;           DROPS THRU TO KEYBOARD AND DISPLAY.

           015A       DISPA:
~~0000~~ (1)→ 015A 03             SCL
      (2)→ 015B C401              LDI   H(CROM)   ;SET ADDRESS OF TABLE.
      ↑    015D 35                XPAH  1
           015E C40B              LDI   L(CROM)
           0160 31                XPAL  1
P2) C602   0161       LOOPD:
  FIRST!   0161 C20C              LD    ADL(2)    ;GET ADDRESS.
           0163 D40F              ANI   0F
           0165 01                XAE
           0166 C180              LD    -128(1)   ;GET SEGMENTS.
           0168 CA04              ST    ADLL(2)   ;SAVE SEG OF ADR LL.
        →  016A C20C              LD    ADL(2)
           016C 1C                SR              ;SHIFT HI DIGIT TO LOW.
           016D 1C                SR
           016E 1C                SR
           016F 1C                SR
           0170 01                XAE
           0171 C180              LD    -128(1)   ;GET SEGMENTS.
           0173 CA05              ST    ADLH(2)
           0175 06                CSA             ;CHECK IF DONE.
           0176 D480              ANI   080
           0178 9809              JZ    DONE
           017A 02                CCL             ;CLEAR FLAG.
           017B C400              LDI   0
           017D CA03              ST    D4(2)     ;ZERO DIGIT 4.
           017F C602              LD    @2(2)     ;FIX P2 FOR NEXT LOOP.
           0181 90DE              JMP   LOOPD
           0183       DONE:
           0183 C6FE              LD    @-2(2)    ;FIX P2.
                      .PAGE 'DISPLAY AND KEYBOARD INPUT'

                      ;           CALL  XPPC  3

                      ;           JMP COMMAND IN A GO = 6,MEM = 7,TERM = 3
                      ;                    IN E GO = 22,MEM = 23,TERM = 27.
                      ;           NUMBER RETURN HEX NUMBER IN E REG.

                      ;           ABORT KEY GOES TO ABORT.
                      ;           ALL REGISTERS ARE USED.

                      ;           P2 MUST POINT TO RAM. ADDRESS MUST BE XXX0.

                      ;           TO RE-EXECUTE ROUTINE DO XPPC3.


           0185       KYBD:
           0185 C400              LDI   0         ;ZERO CHAR.
```

```
0187 CA0B              ST    CHAR(2)
0189 C40D              LDI   H(DISP)     ;SET DISPLAY ADDRESS.
018B 35                XPAH  1
018C         OFF:
018C C4FF              LDI   -1          ;SET ROW/DIGIT ADDRESS.
018E CA10              ST    ROW(2)      ;SAVE ROW COUNTER.
0190 C40A              LDI   10          ;SET ROW COUNT.
0192 CA09              ST    CNT(2)
0194 C400              LDI   0
0196 CA0A              ST    PUSHED(2)   ;ZERO KEYBOARD INPUT.
0198 31                XPAL  1           ;SET DISP ADDRESS LOW.
0199         LOOP:
0199 AA10              ILD   ROW(2)      ;UP DATE ROW ADDRESS.
019B 01                XAE
019C C280              LD    -128(2)     ;GET SEGMENT.
019E C980              ST    -128(1)     ;SEND IT.
01A0 8F00              DLY   0           ;DELAY FOR DISPLAY.
01A2 C180              LD    -128(1)     ;GET KEYBOARD INPUT.
01A4 E4FF              XRI   OFF         ;CHECK IF PUSHED.
01A6 9C4C              JNZ   KEY         ;JUMP IF PUSHED.
01A8         BACK:
01A8 BA09              DLD   CNT(2)      ;CHECK IF DONE.
01AA 9CED              JNZ   LOOP        ;NO IF JUMP.
01AC C20A              LD    PUSHED(2)   ;CHECK IF KEY.
01AE 980A              JZ    CKMORE
01B0 C20B              LD    CHAR(2)     ;WAS THERE A CHAR?
01B2 9CD8              JNZ   OFF         ;YES WAIT FOR RELEASE.
01B4 C20A              LD    PUSHED(2)   ;NO SET CHAR.
01B6 CA0B              ST    CHAR(2)
01B8 90D2              JMP   OFF
01BA         CKMORE:
01BA C20B              LD    CHAR(2)     ;CHECK IF THERE WAS A CHAR.
01BC 98CE              JZ    OFF         ;NO KEEP LOOKING.

                       .PAGE

             ;         COMMAND KEY PROCESSING
01BE         COMMAND:
01BE 01                XAE               ;SAVE CHAR.
01BF 40                LDE               ;GET CHAR.
01C0 D420              ANI   020         ;CHECK FOR COMMAND.
01C2 9C28              JNZ   CMND        ;JUMP IF COMMAND.
01C4 C480              LDI   080         ;FIND NUMBER.
01C6 508F              ANE
01C7 9C1B              JNZ   LT7         ;0 TO 7.
01C9 C440              LDI   040
01CB 50                ANE
01CC 9C19              JNZ   N89         ;8 OR 9.
01CE C40F              LDI   OF
01D0 50                ANE
01D1 F407              ADI   7           ;MAKE OFF SET TO TABLE.
01D3 01                XAE               ;PUT OFF SET AWAY.
```

```
01D4  C080            LD      -128(0)     ;GET NUMBER.
01D6          KEYRTN:
01D6  01              XAE                 ;SAVE IN E.
01D7  C702            LD      @2(3)       ;FIX RETURN.
01D9  3F              XPPC    3           ;RETURN.
01DA  90A9            JMP     KYBD        ;ALLOWS XPPC P3 TO RETURN.

01DC  0A0B           .BYTE 0A.0B,0C,0D,0,0,0E,0F
01DE  0C0D
01E0  0000
01E2  0E0F
01E4          LT7:
01E4  60              XRE                 ;KEEP LOW DIGIT.
01E5  90EF            JMP     KEYRTN
01E7          N89:
01E7  60              XRE                          ;GET LOW.
01E8  F408            ADI        08                ;MAKE DIGIT 8 OR 9.
01EA  90EA            JMP        KEYRTN

                      .PAGE
01EC          CMND:
01EC  60              XRE
01ED  E404            XRI        04                ;CHECK IF ABORT.
01EF  9808            JZ         ABRT              ;ABORT.
01F1  3F              XPPC       3                 ;IN E 23=MEM,22=GO,27=TERM
                                                   ;IN A 7=MEM,6=GO,3=TERM.
01F2  9091            JMP        KYBD              ;ALLOWS JUST A XPPC P3 TO
                                                   ;RETURN.

01F4          KEY:
01F4  58              ORE                          ;MAKE CHAR.
01F5  CA0A            ST         PUSHED(2)         ;SAVE CHAR.
01F7  90AF            JMP        BACK

01F9          ABRT:
01F9  C400            LDI        H(ABORT)
01FB  37              XPAH       3
01FC  C442            LDI        L(ABORT)-1
01FE  33              XPAL       3
01FF  3F              XPPC       3                 ;GO TO ABORT
      0000            .END
```

Mathematical

The mathematical subroutines all take their arguments relative to the pointer register P2. Pointer P3 is the subroutine calling register. All of these routines may be repeated without reloading P3 after the first call.

The simplest way to test them out is to get the monitor to load P2 from memory for you by storing the address required at 0FFB and 0FFC. The arguments should then be entered at successive locations starting at this address, as specified in the diagram labelled 'Stack Usage' at the top of each subroutine. The results are similarly returned relative to P2. To illustrate this procedure the following sequence sets up the multiply subroutine to multiply X'FF by X'FF:

```
0FFB 0F    ; Sets P2H
0FFC 80    ; Sets P2L

0F80 FF    ; Multiplicand
0F81 FF    ; Multiplier
```

The program is entered at 0F50 and the result, X'FE01 in this example, will be put into locations 0F82 and 0F83. A similar process can be followed to use the other mathematical routines.

'Multiply' gives the 16-bit unsigned product of two 8-bit unsigned numbers.

e.g. A = X'FF (255)
B = X'FF (255)
RESULT = X'FE01 (65025).

'Divide' gives the 16-bit unsigned quotient and 8-bit remainder of a 16-bit unsigned dividend divided by an 8-bit unsigned divisor.

e.g. DIVISOR = X'05 (5)
DIVIDEND = X'5768 (22376)
QUOTIENT = X'117B (4475)
REMAINDER = X'01 (1).

'Square Root' gives the 8-bit integer part of the square root of a 16-bit unsigned number. It uses the relation:

$(n + 1)^2 - n^2 = 2n + 1$,

and subtracts as many successive values of $2n + 1$ as possible from the number, thus obtaining n.

e.g. NUMBER = X'D5F6 (54774)
ROOT = X'EA (234).

'Greatest Common Divisor' uses Euclid's algorithm to find GCD of two 16-bit unsigned numbers; i.e. the largest number which will exactly divide them both. If they are coprime the result is 1.

e.g. A = X'FFCE (65486 = 478 × 137)
B = X'59C5 (23701 = 173 × 137)
GCD = X'89 (137).

Multiply

```
                ; Multiplies two unsigned 8-bit numbers
                ; (Relocatable)
                ;
                ; Stack usage:
                ;          REL:      ENTRY:   USE:        RETURN:
                ;           -1                Temp
                ;(P2)->      0       A        A           A
                ;            1       B        B           B
                ;            2                Result (H)  Result (H)
                ;            3                Result (L)  Result (L)
                ;
0000            A          =         0
0001            B          =         1
FFFF            Temp       =         -1
0002            RH         =         2
0003            RL         =         3
                ;

0000                                 .=0F50
0F50    C408    Mult:      LDI       8
0F52    CAFF               ST        Temp (2)
0F54    C400               LDI       0
0F56    CA02               ST        RH(2)
0F58    CA03               ST        RL(2)
0F5A    C201    Nbit:      LD        B(2)
0F5C    02                 CCL
0F5D    1E                 RR
0F5E    CA01               ST        B(2)
0F60    9413               JP        Clear
0F62    C202               LD        RH(2)
0F64    F200               ADD       A(2)
0F66    IF      Shift:     RRL
0F67    CA02               ST        RH(2)
0F69    C203               LD        RL(2)
0F6B    IF                 RRL
0F6C    CA03               ST        RL(2)
0F6E    BAFF               DLD       Temp(2)
0F70    9CE8               JNZ       Nbit
0F72    3F                 XPPC      3
0F73    90DB               JMP       Mult
0F75    C202    Clear:     LD        RH(2)
0F77    90ED               JMP       Shift
                           ;
        0000                         .END
```

Divide

```
                    ; Divides an unsigned 16-bit number by
                    ; an unsigned 8-bit number giving
                    ; 16-bit quotient and 8-bit remainder.
                    ; (Relocatable)
                    ;
                    ; Stack usage:
                    ;        REL:   ENTRY:       USE:          RETURN:
                    ;        -1                  Quotient(l)
                    ;(P2)->   0     Divisor                    Quotient(H)
                    ;        +1     Dividend(H)                Quotient(L)
                    ;        +2     Dividend(L)                Remainder
                    ;
      FFFF          Quot     =      -1
      0000          DSOR     =      0
      0001          DNDH     =      1
      0002          DNDL     =      2
                    ;
0000                         .=0F80
0F80  C200          Div:     LD     DSOR(2)
0F82  01                     XAE
0F83  C400                   LDI    0
0F85  CA00                   ST     DSOR(2)   ;Now Quotient(H)

0F87  CAFF                   ST     Quot(2)   ;Quotient(L)
0F89  C201          Subh:    LD     DNDH(2)
0F8B  03                     SCL
0F8C  78                     CAE
0F8D  CA01                   ST     DNDH(2)
0F8F  1D                     SRL
0F90  9404                   JP     Stoph
0F92  AA00                   ILD    DSOR(2)
0F94  90F3                   JMP    Subh
0F96  C201          Stoph:   LD     DNDH(2)
0F98  70                     ADE              ;Carry is clear
0F99  CA01                   ST     DNDH(2)   ;Undo damage
0F9B  C202          Subl:    LD     DNDL(2)
0F9D  03                     SCL
0F9E  78                     CAE
0F9F  CA02                   ST     DNDL(2)
0FA1  C201                   LD     DNDH(2)
0FA3  FC00                   CAI    0
0FA5  CA01                   ST     DNDH(2)
0FA7  1D                     SRL
0FA8  9404                   JP     Stopl
0FAA  AAFF                   ILD    Quot (2)
0FAC  90ED                   JMP    Subl
0FAE  C202          Stopl:   LD     DNDL(2)
0FB0  70                     ADE
0FB1  CA02                   ST     DNDL(2)   ;Remainder
0FB3  C2FF                   LD     Quot(2)
0FB5  CA01                   ST     DNDH(2)
```

```
0FB7    3F                XPPC    3          ;Return
0FB8    90C6              JMP     Div

        0000              END
```

Square Root

```
                  ; Gives square root of 16-bit unsigned number
                  ; Integer part only. (Relocatable).
                  ;
                  ; Stack usage:
                  ;           REL:    ENTRY:      USE:    RETURN:
                  ;           -1                  Temp
                  ;(P2)->      0      Number(H)           Root(H)
                  ;           +1      Number(L)           Root(L)
                  ;
        0000      HI          =       0
        0001      LO          =       1
        FFFF      Temp        =       -1
                  ;
        0000                  .=0F20
0F20    C400      SQRT:       LDI     X'00
0F22    CAFF                  ST      Temp(2)


0F24    03        Loop:       SCL
0F25    BAFF                  DLD     Temp(2)
0F27    F2FF                  ADD     Temp(2)
0F29    01                    XAE
0F2A    C4FE                  LDI     X'FE
0F2C    F400                  ADI     X'00
0F2E    01                    XAE
0F2F    F201                  ADD     LO(2)
0F31    CA01                  ST      LO(2)
0F33    40                    LDE
0F34    F200                  ADD     HI(2)
0F36    CA00                  ST      HI(2)
0F38    ID                    SRL
0F39    9402                  JP      EXIT
0F3B    90E7                  JMP     LOOP
0F3D    C400      Exit:       LDI     X'00
0F3F    CA00                  ST      HI(2)
0F41    FAFF                  CAD     Temp(2)
0F43    CA01                  ST      LO(2)
0F45    3F                    XPPC    3          ;Return
0F46    90D8                  JMP     SQRT       ;For Repeat
                  ;
0F48                          .=0FFB
                  ;
0FFB    0F80                  .DBYTE  0F80       ;P2-> Number

        0000                  .END
```

Greatest Common Divisor

```
                  ; Finds Greatest Common Divisor of two
                  ; 16-bit unsigned numbers
                  ; uses Euclid's Algorithm. (Relocatable).
                  ;
                  ; Stack usage:
                  ;         REL:    ENTRY:  USE:    RETURN:
                  ;(P2)->   0       A(H)    A(H)    0
                  ;         1       A(L)    A(L)    0
                  ;         2       B(H)    B(H)    GCD(H)
                  ;         3       B(L)    B(L)    GCD(L)
                  ;
        0000      AH        =       0
        0001      AL        =       1
        0002      BH        =       2
        0003      BL        =       3
                  ;
0000                        .=0F20
0F20    03        GCD:      SCL
0F21    C203                LD      BL(2)
0F23    FA01                CAD     AL(2)
0F25    CA03                ST      BL(2)
0F27    01                  XAE
0F28    C202                LD      BH(2)
0F2A    FA00                CAD     AH(2)
0F2C    CA02                ST      BH(2)
0F2E    1D                  SRL             ; Put carry in top bit
0F2F    9402                JP      Swap
0F31    90ED                JMP     GCD     ;Subtract again
0F33    02        Swap:     CCL
0F34    C201                LD      AL(2)
0F36    01                  XAE
0F37    70                  ADE
0F38    CA01                ST      AL(2)
0F3A    40                  LDE
0F3B    CA03                ST      BL(2)
0F3D    C200                LD      AH(2)
0F3F    01                  XAE
0F40    C202                LD      BH(2)
0F42    70                  ADE
0F43    CA00                ST      AH(2)
0F45    01                  XAE
0F46    CA02                ST      BH(2)
0F48    40                  LDE             ;Get new AH(2)
0F49    DA01                OR      AL(2)   ;OR with new AL(2)
0F4B    9CD3                JNZ     GCD     ;Not finished yet
0F4D    3F                  XPPC    3       ;Return
0F4E    90D0                JMP     GCD     ;For repeat run

        0000                .END
```

Electronic

'Pulse Delay' uses a block of memory locations as a long shift-register, shifting bits in at the serial input SIN and out from the serial output SOUT. By varying the delay constants the input waveform can be delayed by up to several seconds, though for a fixed block of memory the resolution of the delay chain obviously decreases with increased delay.

With the program as shown the shift-register uses the 128 locations 0F80 to 0FFF, thus providing a delay of 1024 bits.

The **'Digital Alarm Clock'** gives a continuously changing display of the time in hours, minutes and seconds. In addition, when the alarm time stored in memory tallies with the actual time the flag outputs are taken high. The time can be set in locations 0F16, 0F17, and 0F18, and the alarm time is stored in locations 0F12, 0F13, and 0F14.

The program should be executed at 0F20.

The program depends for its timing on the execution time of the main loop of the program, which is executed 98 times a second, so this is padded out to exactly 1/98th of a second with a delay instruction. The delay constants at 0F7F and 0F81 should be adjusted to give the correct timing.

'Random Noise' generates a pseudo-random sequence of 2^{15}-1 or 65535 bits at the flag outputs. If one flag output is connected to an amplifier the sequence sounds like random noise. Alternatively, by converting the program to a subroutine to return one bit it could be used to generate random coin-tosses for games and simulations. Note that the locations 0F1E and 0F1F must not contain 00 for the sequence to start.

Pulse Delay

```
                          ; Pulse delayed by 1024 bit-times.
                          ; (Relocatable). Uses serial in/out.
                          ;
0000                                .=0F1F
0F1F                      Bits:     .=.+1                 ;bit counter
                          ;
0F20      C40F            Enter:    LDI        H(Scrat)
0F22      35                        XPAH       1
0F23      C480                      LDI        L(Scrat)
0F25      31              Next:     XPAL       1
0F26      C408                      LDI        8
0F28      C8F6                      ST         Bits
0F2A      C100                      LD         (1)        ;Get old byte
0F2C      01                        XAE                   ;Exchange
0F2D      CD01                      ST         @+1(1)     ;Put back new byte
0F2F      19              Output:   SIO                   ;Serial I/O
0F30      C400                      LDI        TC1
0F32      8F04                      DLY        TC2        ;Delay bits
0F34      B8EA                      DLD        Bits
0F36      9CF7                      JNZ        Output
0F38      31                        XPAL       1          ;P1 = 0D00 Yet?
```

```
0F39   9CEA            JNZ     Next
0F3B   90E3            JMP     Enter
                ;
       0000     TC1    =       0          ;Bit-time
       0004     TC2    =       4          ;Delay constants
                ;
       0F80     Scrat  =       0F80       ;Start of scratch area
       0000            .END
```

Digital Alarm Clock

```
                ;Outputs are held on when alarm
                ;time = Actual time,  i.e. for one sec.
                ;Enter at 0F20
       010B     Crom     =       010B       ;Segment table
       0D00     Disp     =       0D00       ;Display address
       0F00     Ram      =       0F00
       0F10     Row      =       Ram + 010
0000                     . = 0F12
0F12                     . = . + 1          ;Alarm time:hours
0F13                     . = . + 1          ;Minutes
0F14                     . = . + 1          ;Seconds
0F15                     . = . + 1          ;Not used
0F16            Time:    . = . + 4          ;Actual time
0F1A   76                .BYTE   076        ;Excess: Hours
0F1B   40                .BYTE   040        ;Minutes
0F1C   40                .BYTE   040        ;seconds
0F1D   20       Speed:   .BYTE   002        ;Speed
0F1E                     . = 0F20
0F20   C401     Clock:   LDI     H(Crom)
0F22   37                XPAH    3
0F23   C40B              LDI     L(Crom)
0F25   33                XPAL    3
0F26   C40D     New:     LDI     H(Disp)
0F28   36                XPAH    2
0F29   C40D              LDI     L (Disp)  + 0D
0F2B   32                XPAL    2
0F2C   C40F              LDI     H(Time)
0F2E   35                XPAH    1
0F2F   C41A              LDI     L(Time) + 4
0F31   31                XPAL    1
0F32   03                SCL
0F33   C405              LDI     5          ;Loop count
.0F35  C8DA              ST      Row
0F37   C5FF     Again:   LD      @—1(1)
0F39   EC00              DAI     0
0F3B   C900              ST      (1)
0F3D   E904              DAD     +4(1)
0F3F   9804              JZ      Cs
0F41   9802              JZ      Cs         ;Equalize paths
0F43   9002              JMP     Cont
0F45   C900     Cs:      ST      (1)
```

```
0F47   C100   Cont:    LD     (1)
0F49   D40F            ANI    0F
0F4B   01              XAE
0F4C   C380            LD     -128(3) ;Get segments
0F4E   CE01            ST     @+1(2)  ;Write to display
0F50   C440            LDI    040
0F52   8F00            DLY    00      ;Equalize display
0F54   C100            LD     (1)
0F56   1C              SR
0F57   1C              SR
0F58   1C              SR
0F59   1C              SR
0F5A   01              XAE
0F5B   C380            LD     -128(3)
0F5D   CE02            ST     @+2(2)  ;Leave a gap
0F5F   B8B0            DLD    Row
0F61   9CD4            JNZ    Again
0F63   C403            LDI    3
0F65   C8AA            ST     Row     ;Digit count
0F67   C400            LDI    0
0F69   01              XAE
0F6A   C5FF   Loop:    LD     @-1(1)
0F6C   E104            XOR    +4(1)   ;Same time?
0F6E   58              ORE
0F6F   01              XAE
0F70   B89F            DLD    Row
0F72   9CF6            JNZ    Loop
0F74   01              XAE
0F75   9803            JZ     Alarm   ;Times tally
0F77   40              LDE
0F78   9003            JMP    Contin
0F7A   C407   Alarm:   LDI    07      ;All flags on
0F7C   08              NOP            ;Pad out path
0F7D   07     Contin:  CAS            ;Output to flags
0F7E   C402            LDI    02      ;Pad out loop to
0F80   8F11            DLY    011     ;1/(100-speed) secs.
0F82   90A2            JMP    New

       0000            .END
```

Random Noise

```
                ; Relocatable
                ; Generates sequence 2↑15 bits long
                ;
                        .=0F1E
0F1E            Line:   .=.+2           ;For random number
                ;                       ;Must not be zero
0F20   C0FD     Noise:  LD      Line
0F22   1F               RRL
0F23   C8FA             ST      Line
0F25   C0F9             LD      Line+1
```

```
OF 27   1F        RRL
OF 28   C8F6      ST      Line + 1
OF 2A   02        CCL                ;Ex-or of bits 1 and 2
OF 2B   F402      ADI     02         ;In bit 2
OF 2D   1E        RR                 ;Rotate bit 2 to
OF 2E   1E        RR                 ;Bit 7
OF 2F   1E        RR
OF 30   D487      ANI     087        ;Put it in carry and
OF 32   07        CAS                ;Update flags
OF 33   90EB      JMP     Noise

        0000      .END
```

System

'Decimal to Hex' displays in hex the decimal number entered in at the keyboard as it is being entered. Negative numbers can be entered too, prefixed by 'MEM'.

e.g. 'MEM' '1' '5' '7' displays 'FF63'

'TERM' clears the display ready for a new number entry.

'Relocator' will move up to 256 bytes at a time from any start address to any destination address.

These two addresses and the number of bytes to be moved are specified in the 5 locations before the program. Since the source program and destination area may overlap, the order in which bytes are transferred is critical to avoid overwriting data not yet transferred, and so the program tests for this. The program should be executed at 0F20.

Any of the programs marked relocatable can be moved, without alteration, to a different start address and they will execute in exactly the same manner.

Serial Data Transfers

This section describes a method of serial data input/output (I/O) data transfer using the Extension Register. All data I/O is under direct software control with data transfer rates between 110 baud and 9600 baud selectable via software modification.

Data Output

Data to be output is placed in the Extension Register and shifted out through the SOUT Port using the Serial Input/Output Instruction (SIO). The Delay Instruction (DLY), in turn, creates the necessary delay to achieve the proper output baud rate. This produces a TTL-level data stream which can be used as is or can be level-shifted to an RS-232C level. Numerous circuits are available for level shifting. As an example, either a DS1488 or an operational amplifier can be used. Inversion of the data stream, if needed, can be done either before the signal is converted or by the level shifter itself.

Data Input

Data input is received in much the same way as data is output. The Start Bit is sensed at the SIN Port and then received using the SIO Instruction and the DLY Instruction. After the Start Bit is received, a delay into the middle of the bit-time is executed. the data is then sensed at each full bit-time (the middle of the bit) until all data bits are received. If the data is at an RS-232C level, it must be shifted to a TTL level.

This can be done with either a DS 1489 or an operational amplifier. If inversion if the data is necessary, it should be done before it is presented to the SIN Port.

Timing Considerations

Serial data transmission rates can be varied by simply changing the delay constants in each of the programs. Table 1 contains the delay constants needed for the various input baud rates. Table 2 contains the delay constants needed for the various output baud rates.

Baud Rate	Bit Time	HBTF	HBTC	BTF	BTC
110	9.09 ms	X'C3	X'8	X'92	X'11
300	3.33 ms	X'29	X'3	X'5E	X'6
600	1.67 ms	X'8A	X'1	X'20	X'3
1200	0.833ms	X'BB	X'0	X'81	X'1
2400	0.417ms	X'52	X'0	X'B2	X'0
4800	0.208ms	X'1F	X'0	X'4A	X'0
6400	0.156ms	X'12	X'0	X'30	X'0
9600	0.104ms	X'5	X'0	X'16	X'0

Table 1. Input Delay Constants (4 MHz SC/MP-II)

Baud Rate	Bit Time	BTF1	BTF2	BTC
110	9.09 ms	X'91	X'86	X'11
300	3.33 ms	X'5E	X'53	X'6
600	1.67 ms	X'1F	X'14	X'3
1200	0.833 ms	X'81	X'76	X'1
2400	0.417 ms	X'B2	X'A7	X'0
4800	0.208 ms	X'49	X'3E	X'0
6400	0.156 ms	X'2F	X'24	X'0
9600	0.104 ms	X'15	X'A	X'0

Table 2. Output Delay Constants (4 MHz SC/MP-II)

NOTES:

1. The Serial Data Output routine requires that the bit-count (BITCNT) in the program be set to the total number of data bits and stop bits to be used per character.

2. Two stop bits are needed for the 110 baud rate; all other baud rates need only one stop bit.

Decimal to Hex

```
                  ; Converts decimal number entered at
                  ; keyboard to hex and displays result
                  ;
                  ; 'MEM' = minus, 'TERM' clears display
                  ; (Relocatable)
        000C      ADL      =        0C
        000E      ADH      =        0E
        0F00      Ram      =        0F00
        015A      Dispa    =        015A
        0011      Count    =        011
        0012      Minus    =        012
        0013      Ltemp    =        013
                  ;
0000                       . = 0F50
0F50    C400      Dhex:    LDI      0
0F52    CA12               ST       Minus(2)
0F54    CA0E               ST       ADH(2)
0F56    CA0C               ST       ADL(2)
0F58    C401      Disp:    LDI      H(Dispa)
0F5A    37                 XPAH     3
0F5B    C459               LDI      L(Dispa)-1
0F5D    33                 XPAL     3
0F5E    3F                 XPPC     3
0F5F    9028               JMP      Comd       ;Command key
0F61    C40A               LDI      10         ;Number in extension
0F63    CA11               ST       Count(2)   ;Multiply by 10
0F65    03                 SCL
0F66    C212               LD       Minus(2)
0F68    01                 XAE
0F69    60                 XRE
0F6A    78                 CAE
0F6B    01                 XAE
0F6C    40                 LDE                 ;Same as: LDI 0
0F6D    78                 CAE                 ;         CAD 0
0F6E    01                 XAE
0F6F    9002               JMP      Digit
0F71    C213      Addd:    LD       Ltemp(2)   ;Low byte of product
0F73    02        Digit:   CCL
0F74    F20C               ADD      ADL(2)
0F76    CA13               ST       Ltemp(2)
0F78    40                 LDE                 ;High byte of product
0F79    F20E               ADD      ADH(2)
0F7B    01                 XAE                 ;Put back
0F7C    BA11               DLD      Count(2)
0F7E    9CF1               JNZ      Addd
```

```
0F80    40                  LDE
0F81    CA0E                ST        ADH(2)
0F83    C213                LD        Ltemp(2)
0F85    CA0C                ST        ADL(2)
0F87    90CF                JMP       Disp        ;Display result
0F89    E403      Comd:     XRI       3           ;'TERM'?
0F8B    98C3                JZ        Dhex        ;Restart if so
0F8D    C4FF                LDI       X'FF        ;Must be 'MEM'
0F8F    CA12                ST        Minus(2)
0F91    90C5                JMP       Disp
                  ;
0F93                        . = 0FFB
0FFB    0F00                .DBYTE    Ram         ;Set P2-> Ram
                  ;
        0000                .END
```

Relocator

```
                  ;Moves block of memory
                  ;'From' = source start address
                  ;'To' = destination start address
                  ;'Length' = No of bytes
                  ;(Relocatable)
                  ;
        FF80      E         =         -128        ;Extension as offset
0000                        . = 0F1B
                  ;
0F1B              From:     . = . + 2
0F1D              To:       . = . + 2
0F1F              Length:   . = . + 1
                  ;
0F20    C400      Entry:    LDI       0
0F22    01                  XAE
0F23    03                  SCL
0F24    C0F9                LD        To + 1
0F26    F8F5                CAD       From + 1
0F28    C0F4                LD        To
0F2A    F8F0                CAD       From
0F2C    1D                  SRL
0F2D    9403                JP        Fgt         ;'From' greater than 'To'
0F2F    C0EF                LD        Length      ;Start from end
0F31    01                  XAE
0F32    02        Fgt:      CCL
0F33    C0E8                LD        From + 1
0F35    70                  ADE
0F36    31                  XPAL      1
0F27    C0E3                LD        From
0F39    F400                ADI       0
0F3B    35                  XPAH      1
0F3C    02                  CCL
0F3D    C0E0                LD        To + 1
0F3F    70                  ADE
```

```
0F40   32                   XPAL    2
0F41   C0DB                 LD      To
0F43   F400                 ADI     0
0F45   36                   XPAH    2
0F46   02                   CCL
0F47   40                   LDE
0F48   9C02                 JNZ     Up
0F4A   C402                 LDI     2
0F4C   78          Up:      CAE             ;i.e. subtract 1
0F4D   01                   XAE             ;Put it in ext.
0F4E   C580        Move:    LD      @E(1)
0F50   CE80                 ST      @E(2)   ;Move byte
0F52   B8CC                 DLD     Length
0F54   9CF8                 JNZ     Move
0F56   3F                   XPPC    3       ;Return

       0000                 .END
```

Serial Data Input

```
; Routine is called with a ''XPPC3'' instruction

; Data is received through the serial I/O Port.

; Before executing routine, Pointer P2 should point
; to one available location in R/W memory for a
; counter.
; On return from routine, data received will be in the
; Accumulator and the Extension Register.

; Delay Constants, user defined for desired Baud rate.
; The following example is for 1200 Baud:

      00BB  HBTF   =     0BB    ; Half Bit time, Fine
      0000  HBTC   =     0      ; Half Bit time, Coarse
      0081  BTF    =     081    ; Full Bit Time, Fine
      0001  BTC    =     01     ; Full Bit time, Coarse

            Search:
0000  C408         LDI   08     ; Initialize Loop Counter
0002  CA00         ST    (2)    ; Save in memory
            Again:
0004  C400         LDI   0      ;Clear Accumulator
0006  01           XAE          ; Clear E. Reg.
0007  19           S10          ;Look for Start Bit
0008  40           LDE          ; Bring into Acc.
0009  9CF9         JNZ   Again  ; If not zero, look again
000B  C4BB         LDI   HBTF   ; Load Acc Half Bit time
000D  8F00         DLY   HBTC   ; Delay Half Bit time
000F  19           SIO          ; Check Input again to
0010  01           XAE          ; be sure of Start Bit
```

```
0011  9CF1            JNZ    Again    ; If not zero, was not
0013  C400            LDI    0        ; start B
0015  01              XAE
               Loop:
0016  C481            LDI    BTF      ; Load Bit time Fine
0018  8F01            DLY    BTC      ; Delay one Bit time
001A  19              SIO             ; Shift in Data Bit
001B  BA00            DLD    (2)      ; decrement loop counter
001D  9CF7            JNZ    Loop     ;Test for done
001F  40              LDE             ; Done, put data in acc.
0020  3F              XPPC   P3

      0000            END
```

Serial Data Output

```
      ; Routine is called with a ''XPPC3'' instruction.

      ; Data is transmitted through Serial I/O Port.

      ; Before executing subroutine, pointer P2 should
      ; point to one available byte of R/W memory for a
      ; counter.
      ; Upon entry, character to be transmitted must be in
      ; the accumulator.

      ; Delay constants, user defined for desired baud rate.
      ; The following example is for 1200 baud:

0081  BTF1     =       081      ; Bit time Fine, first loop
0076  BTF2     =       076      ; Bit time Fine, second loop
0001  BTC      =       01       ; Full Bit time, Coarse
               ; Character Bit-count. This should be set for the
               ; desired number of Data Bits and stop Bits.

      0009     BITCNT  =        9        ; 8 data and 1 Stop Bit

               Start:
0000  01              XAE             ; Save data in E. Reg.
0001  C400            LDI    0        ; Clear acc.
0003  01              XAE             ; Put data in acc, clear E.
0004  19              SIO             ; Send Start Bit
0005  01              XAE             ; Put data in E. Reg.
0006  C481            LDI    BTF1     ; Load Bit time Fine
0008  8F01            DLY    BTC      ; Wait one Bit time
000A  C409            LDI    BITCNT   ; Set loop count for data
000C  CA00            ST     (2)      ; and Stop Bit(s). Save
               Send:                  ; in count.
000E  19              SIO             ; Send Bit
000F  40              LDE
0010  DC80            ORI    080      ; Set last Bit to 1
0012  01              XAE             ; Put back in E. Reg.
0013  C476            LDI    BTF2     ; Load Bit time Fine
```

```
0015 8F01          DLY    BTC     ; Delay one Bit time
0017 BA00          DLD    (2)     ; decrement Bit counter
0019 9CF3          JNZ    Send    ; If not done, loop back
001B 3F            XPPC    3      ; otherwise, return

     0000         .END
```

Games

The first two games are real-time simulations which provide a test of skill, and they can be adjusted in difficulty to suit the player's ability. The last two games are both tests of clear thinking and logical reasoning, and in the last one you are pitted against the microprocessor which tries to win.

'Moon Landing' simulates the landing of a spacecraft on the moon. The displays represent the control panel and give a continuously changing readout of altitude (3 digits), rate of descent (2 digits), and fuel remaining (1 digit). The object of the game is to touch down gently; i.e. to reach zero altitude with zero rate of descent. To achieve this you have control over the thrust of the rockets: the keys 1 to 7 set the thrust to the corresponding strength, but the greater the thrust the higher the rate of consumption of fuel. When the fuel runs out an 'F' is displayed in the fuel gauge, and the spacecraft will plummet to the ground under the force of gravity.

On reaching the moon's surface the display will freeze showing the velocity with which you hit the surface if you crashed, and the fuel remaining. Pressing 'TERM' will start a new landing.

The program should be entered at OF52.

The speed of the game is determined by the delay constants at OF38 and OF3A. The values given are suitable for a 1 MHz clock and they should be increased in proportion for higher clock rates. The initial values for the altitude, velocity, and fuel parameters are stored in memory at OF14 to OF1F and these can be altered to change the game.

'Duck Shoot' simulates ducks flying across the skyline. At first there is one duck, and it can be shot by hitting the key corresponding to its position: 7 = leftmost display, 0 = rightmost display. If you score a hit the duck will disappear; if you miss however, another duck will appear to add to you task.

The counter at OF1D varies the speed of flight and can be increased to make the game easier.

'Mastermind' consists of trying to deduce a 'code' chosen by the machine. The code consists of four decimal digits, and pressing 'TERM' followed by'MEM' causes the machine to choose a new code. The player makes guesses at the code which are entered at the keyboard. Pressing 'GO' then causes the machine to reveal two pieces of information, which are displayed as two digits:

(1) The number of digits in the guess which are correct and in the right position, (known as 'Bulls') and

(2) the number of digits correct but in the wrong position, (known as 'Cows').

For example, suppose that the machine's code was '6678'. The following guesses would then score as shown:

1234	00	1278	20
7812	02	7687	12

Subsequent guesses are entered in a similar way, and the player tries to deduce the code in as few attempts as possible.

'Silver Dollar Game' is traditionally played with a number of coins which are moved by the players in one direction along a line of squares. In his turn, a player must move a coin to the right across as many unoccupied squares as he wishes. The player first unable to move—when all the coins have reached the right-hand end of the line—loses, and the other player takes the coins!

In this version of the game the coins are represented by vertical bars moving along a dashed line. There are five coins numbered, from right to left, 1 to 5. The player makes his move by pressing the key corresponding to the number of the coin he wishes to move, and each press moves the coin one square along to the right. The machine plays against you, and pressing 'MEM' causes it to make its move. Note that the machine will refuse to move in its turn unless you have made a legal move in your turn. 'TERM' starts a new game.
The program should be entered at 0F28.

The machine allows you to take first move and it is possible to win from the starting position given, though this is quite difficult. The five numbers in locations 0F13 to 0F17 determine the starting positions of each coin and these can be altered to any other values in the range 00 to 0F provided they are in ascending order.

Moon Landing

```
                ; Land a rocket on the moon
                ; Display shows altitude-velocity-fuel
                ; Keys 1-7 control the thrust
                ;
      0005      Grav     =       5         ;Force of gravity
      0D00      Disp     =       0D00      ;Display address
      010B      Crom     =       010B      ;Segment table
      FF80      E        =       -128      ;Extension as offset
      FFE3      Row      =       0F03-Ret
      FFE4      Count    =       0F04-Ret
                ;Variables
0000                     .=0F05
0F05            Save:    .=.+1
0F06            H1:      .=.+1
0F07            L1:      .=.+1
0F08            Alt:     .=.+3             ;Altitude
0F0B            Vel:     .=.+3             ;Velocity
0F0E            Accn:    .=.+2             ;Acceleration
0F10            Thr:     .=.+2             ;Thrust
0F12            Fuel:    .=.+2             ;Fuel left
                ;Original values
0F14  08        Init:    BYTE    08,050,0;Altitude = 850
      50
      00
0F17  99                 .BYTE   099,080,0;Velocity = -20
      80
      00
0F1A  99                 .BYTE   099,098  ;Acceleration = -2
      98
0F1C  00                 .BYTE   0,02      ;Thrust = 2
      02
0F1E  58                 .BYTE   058,0     ;Fuel = 5
      00
```

```
                         ;Subroutine to display AC as two digits
0F20    3E        Ret:       XPPC     2          ;P2 contains 0F20
0F21    C8E3                 ST       Save
0F23    C401                 LDI      H(Crom)
0F25    35                   XPAH     1
0F26    C8DF                 ST       H1         ;Run out of pointers
0F28    C40B                 LDI      L(Crom)
0F2A    31                   XPAL     1
0F2B    C8DB                 ST       L1
0F2D    C0D7                 LD       Save
0F2F    02                   CCL
0F30    D40F                 ANI      0F
0F32    01                   XAE
0F33    C180      Loop:      LD       E(1)
0F35    CF01                 ST       @+1(3)
0F37    C400                 LDI      0          ;Delay point
0F39    8F04                 DLY      4          ;Determines speed
0F3B    C0C9                 LD       Save
0F3D    1C                   SR
0F3E    1C                   SR
0F3F    1C                   SR
0F40    1C                   SR
0F41    01                   XAE
0F42    06                   CSA
0F43    03                   SCL
0F44    94ED                 JP       Loop       ;Do it twice
0F46    C400                 LDI      0
0F48    CF01                 ST       @+1(3)     ;Blank between
0F4A    C0BB                 LD       H1         ;Restores P1:
0F4C    35                   XPAH     1
0F4D    C0B9                 LD       L1
0F4F    31                   XPAL     1
0F50    90CE                 JMP      Ret        ;Return
                         ;Main moon-landing program
                         ;Begin execution here.
0F52    C40F      Start:     LDI      H(Init)
0F54    35                   XPAH     1
0F55    C414                 LDI      L(Init)
0F57    31                   XPAL     1
0F58    C40F                 LDI      H(Ret)
0F5A    36                   XPAH     2
0F5B    C420                 LDI      L(Ret)
0F5D    32                   XPAL     2
0F5E    C40C                 LDI      12
0F60    CAE4                 ST       Count(2)
0F62    C10B      Set:       LD       +11(1)
0F64    CDFF                 ST       @-1(1)
0F66    BAE4                 DLD      Count(2)
0F68    9CF8                 JNZ      Set
                         ;Main loop
0F6A    C40C      Again:     LDI      H(Disp)-1
0F6C    37                   XPAH     3
0F6D    C4FF                 LDI      L(Disp)-1
0F6F    33                   XPAL     3
0F70    C401                 LDI      1
0F72    CAE4                 ST       Count(2)
```

```
0F74    C506              LD      @+6(1)     ;P1->Vel+2
0F76    9404              JP      Twice      ;Altitude positive?
0F78    C504              LD      @+4(1)     ,P1->Thr+1
0F7A    9032              JMP     Off        ;Don't update
0F7C    C402    Twice:    LDI     2          ;Update velocity anc
0F7E    CAE3              ST      Row(2)     ;Then altitude....
0F80    02                CCL
0F81    C5FF    Dadd:     LD      @-1(1)
0F83    E902              DAD     +2(1)
0F85    C900              ST      (1)
0F87    BAE3              DLD     Row(2)
0F89    9CF6              JNZ     Dadd
0F8B    C102              LD      +2(1)
0F8D    9402              JP      Pos        ;Gone negative?
0F8F    C499              LDI     X'99
0F91    EDFF    Pos:      DAD     @-1(1)
0F93    C900              ST      (1)
0F95    BAE4              DLD     Count(2)
0F97    94E3              JP      Twice
0F99    C50C              LD      @12(1)     ;P1->Alt
0F9B    AAE3              ILD     Row(2)     ;Row:=1
0F9D    03                SCL
0F9E    C5FF    D sub:    LD      @-1(1)     ;Fuel
0FA0    F9FE              CAD     -2(1)      ;Subtract thrust
0FA2    C900              ST      (1)
0FA4    08                NOP
0FA5    BAE3              DLD     Row(2)
0FA7    94F5              JP      Dsub
0FA9    06                CSA                ;P1->Fuel now
0FAA    9402              JP      Off        ;Fuel run out?
0FAC    9004              JMP     Accns
0FAE    C400    Off:      LDI     0
0FB0    C9FF              ST      -1(1)      ;Zero thrust
0FB2    C1FF    Accns:    LD      -1(1)
0FB4    03                SCL
0FB5    EC94              DAI     099-Grav
0FB7    C9FD              ST      -3(1)      ;Accn+1
0FB9    C499              LDI     X'99
0FBB    EC00              DAI     0
0FBC    C9FC              ST      -4(1)      ;Accn
0FBF    C100    Dispy:    LD      (1)        ;Fuel
0FC1    3E                XPPC    2          ;Display it OK
0FC2    C1F9              LD      -7(1)      ;Vel
0FC4    940A              JP      Posv
0FC6    C499              LDI     X'99
0FC8    03                SCL
0FC9    F9FA              CAD     -6(1)      ;Vel+1
0FCB    03                SCL
0FCC    EC00              DAI     0
0FCE    9002              JMP     STO
0FD0    C1FA    Posv:     LD      -6(1)      ;Vel+1
0FD2    3E      Sto:      XPPC    2          ;Display velocity
0FD3    C1F7              LD      -9(1)      ;Alt+1
```

```
0F2E   C461           LDI    Duck
0F30   9002           JMP    Go
0F32   C400   No:     LDI    0          ;No duck
0F34   C980   Go:     ST     -128(1)    ;E as offset
0F36   8F01           DLY    01         ;Shine digit
0F38   C0D8           LD     Sum
0F3A   9C0E           JNZ    Nok        ;Key already pressed
0F3C   C180           LD     -128(1)    ;Test for key
0F3E   E4FF           XRI    0FF
0F40   9808           JZ     Nok        ;No key
0F42   C8CE           ST     Sum
0F44   C0CA           LD     Row
0F46   E480           XRI    080
0F48   C8C6           ST     Row        ;Change top bit
0F4A   40     Nok:    LDE
0F4B   03             SCL
0F4C   FC01           CAI    1          ;Subtract 1
0F4E   94D6           JP     Ndig       ;Do next digit
0F50   B8BF           DLD    Count
0F52   98C8           JZ     React      ;Start new position
0F54   C407           LDI    7
0F56   90CE           JMP    Ndig       ;Another sweep
       0000           .END
```

Mastermind

```
       0F00   Ram     =      0F00
       0D00   Disp    =      0D00     ;Display address
       010B   Crom    =      010B     ;Hex to segment table
       011B   Adr     =      011B     ;'Make 4 digit address'
       015A   Dispa   =      015A     ;'Address to segments'
              ;       Variables in RAM
       0000   DL      =      0
       0002   DH      =      1
       0004   Adll    =      4
       000C   Adl     =      12
       000E   Adh     =      14
       000F   Ddta    =      15
       0010   Row     =      16
       0011   Next    =      17
       0014   Key     =      20
              ;       Begin at 0FIC
0000                  . = 0FIC
0F1C   C400   Start:  LDI    0
0F1E   C8ED           ST     ADL
0F20   C8ED           ST     ADH
0F22   32             XPAL   2
0F23   C40F           LDI    0F
0F25   36             XPAH   2
              ;       Choose  random   number
0F26   C401           LDI    H(Crom)
0F28   37             XPAH   3
```

```
OF29   C40B              LDI     L(Crom)
OF2B   33                XPAL    3
OF2C   C404   No Key:    LDI     04
OF2E   CA10              ST      Row(2)
OF30   C40F              LDI     H(digits)
OF32   35                XPAH    1
OF33   C414              LDI     L(Digits)
OF35   31                XPAL    1
OF36   03                SCL
OF37   C104   Incr:      LD      +4(1)
OF39   EC90              DAI     090
OF3B   C904              ST      +4(1)
OF3D   D40F              ANI     OF
OF3F   01                XAE
OF40   C380              LD      -128(3)
OF42   CD01              ST      @+1(1)
OF44   BA10              DLD     Row(2)
OF46   9CEF              JNZ     Incr
OF48   C40D              LDI     H(Disp)
OF4A   35                XPAH    1
OF4B   C400              LDI     L(Disp)
OF4D   31                XPAL    1
OF4E   C103              LD      3(1)       ;Key pressed?
OF50   E4FF              XRI     OFF
OF52   98D8              JZ      No key
                      ,  Enter your guess
OF54   C4FF   Clear:     LDI     OFF
OF56   CA0F              ST      Ddta(2)
OF58   C400   Blank:     LDI     0
OF5A   CA00              ST      DL(2)
OF5C   CA01              ST      DH(2)
OF5E   02     Nchar:     CCL
OF5F   C401              LDI     H(Dispa)
OF61   37                XPAH    3
OF62   C459              LDI     L(Dispa)-1
OF64   33                XPAL    3
OF65   3F                XPPC    3          ;Jump to subroutine
OF66   900B              JMP     COMD       ;Command key return
OF68   40                LDE                ;Number key return
OF69   F4F6              ADI     OF6
OF6B.  94F1              JP      Nchar      ;Ignore digits > 9
OF6D   C41A              LDI     L(Adr)-1
OF6F   33                XPAL    3
OF70   3F                XPPC    3
OF71   90E5              JMP     Blank      ;Get next digit
OF73   E403   Comd:      XRI     03         ;term?
OF75   9A1B              JZ      Start(2)   ;If so-new game
OF77   E405              XRI     05         ;Go?
OF79   9CD9              JNZ     Clear      ;Ignore if not
                      ,  Work out answer to guess
OF7B   C40B   Go:        LDI     L(Crom)
OF7D   CA00              ST      DL(2)
OF7F   CA01              ST      DH(2)
OF81   C40F   Bulls:     LDI     H(Key)
```

```
0F83   35                XPAH   1
0F84   C414              LDI    L(Key)
0F86   31                XPAL   1
0F87   C480              LDI    080
0F89   01                XAE
0F8A   C404              LDI    04          ;No. of digits
0F8C   CA11              ST     Next(2)
0F8E   C1F0    Bull 2:   LD     Adll-Key(1)
0F90   E501              XOR    @+1(1)
0F92   9C0C              JNZ    Nobul
0F94   AA01              ILD    DH(2)
0F96   C1FF              LD     —1(1)
0F98   58                ORE                ;Set negative
0F99   C9FF              ST     —1(1)
0F9B   C1EF              LD     Adll-Key-1(1)
0F9D   58                ORE
0F9E   C9EF              ST     Adll-Key-1(1)
0FA0   BA11    Bobul:    DLD    Next(2)
0FA2   9CEA              JNZ    Bull 2
0FA4   C404    Cows:     LDI    04
0FA6   CA11              St     Next(2)     ;P1 points to Key + 4
0FA8   C404    Nerow:    LDI    04
0FAA   CA10              ST     Row(2)
0FAC   C40F              LDI    H(Adll)
0FAE   37                XPAH   3
0FAF   C408              LDI    L(Adll) + 4
0FB1   33                XPAL   3
0FB2   C5FF              LD     @—1(1)
0FB4   940A              JP     Try         ;Already counted as bull?
0FB6   BA11    Nocow:    DLD    Next(2)     ;Yes
0FB8   9CEE              JNZ    Nerow
0FBA   9013              JMP    Finito
0FBC   BA10    Notry:    DLD    Row(2)
0FBE   98F6              JZ     Nocow
0FC0   C100    Try:      LD     (1)
0FC2   E7FF              XOR    @—1(3)      :Same?
0FC4   9CF6              JNZ    Notry
0FC6   AA00              ILD    DL(2)
0FC8   C300              LD     (3)
0FCA   58                ORE
0FCB   CB00              ST     (3)
0FCD   90E7              JMP    Nocow
                       ; Now unset top bits of Key
0FCF   C404    Finito:   LDI    04
0FD1   CA11              ST     Next(2)
0FD3   C100    Unset:    LD     (1)
0FD5   D47F              ANI    07F
0FD7   CD01              ST     @+1(1)
0FD9   BA11              DLD    Next(2)
0FDB   9CF6              JNZ    Unset       ;All done?
```

```
                       ;Set up segments of result
OFDD   C401                   LDI     H(Crom)
OFDF   35                     XPAH    1
OFE0   C200                   LD      DL(2)     ;L(Crom) + Cows
OFE2   31                     XPAL    1
OFE3   C100                   LD      (1)       ;Segments
OFE5   CA00                   ST      DL(2)
OFE7   C201                   LD      DH(2)     ;L(Crom) + Bulls
OFE9   31                     XPAL    1
OFEA   C100                   LD      (1)       ;Segments
OFEC   CA01                   ST      DH(2)
OFEE   C4FF                   LDI     OFF
OFF0   CA0F                   ST      Ddta(2)
OFF2   925D                   JMP     Nchar(2)  ;Display result
                       ;
       0000                   .END
```

Silver Dollar Game

```
                ; Machine plays against you in moving five
                ; 'Silver Dollars' along a track
                ; Player unable to move loses
0000                          =0F12
                ; Starting position: Must be ascending order
0F12    FF      Start:      .BYTE    OFF
0F13    03                  .BYTE    03
0F14    05                  .BYTE    05
0F15    08                  .BYTE    08
0F16    09                  .BYTE    09
0F17    OF                  .BYTE    0
        0F00    Ram         =        0F00
0F18            Pos:        .=.+6              ;Current position
        0024    Count       =        024       ;Ram offsets:
        0025    Key         =        025       ;For key last pressed
        0026    Init        =        026       ;Zero
        0185    Kybd        =        0185      ;In monitor
        FF80    E           =        -128      ;Extension reg.
                ;
                ;Begin execution here
0F1E                        .=0F28
0F28    C40F    Begin:      LDI      H(Ram)
0F2A    36                  XPAH     2
0F2B    C400                LDI      L(Ram)
0F2D    32                  XPAL     2
0F2E    C40F                LDI      H(Pos)
0F30    35                  XPAH     1
0F31    C418                LDI      L(Pos)
0F33    31                  XPAL     1
0F34    C406                LDI      6
0F36    CA24                ST       Count (2)
0F38    C1FA    Setup:      LD       -6(1)     ;Transfer start to pos
0F3A    CD01                ST       @+1(1)
0F3C    BA24                DLD      Count(2)
```

```
0F3E   9CF8                JNZ     setup
0F40   C400     Ymove:     LDI     0          ;You go first!
0F42   CA25                ST      Key(2)     ;Clear key store
                ;Generate display from Pos
0F44   C40F     Disp:      LDI     H(Pos)
0F46   35                  XPAH    1
0F47   C419                LDI     L(Pos) + 1
0F49   31                  XPAL    1
0F4A   C409                LDI     9
0F4C   01       Clear:     XAE                ;Clear Display buffer
0F4D   C408                LDI     08         ;Underline
0F4F   CA80                ST      E(2)
0F51   40                  LDE
0F52   FC01                CAI     1
0F54   94F6                JP      Clear
0F56   C405                LDI     5
0F58   CA24                ST      Count(2)
0F5A   C501     Npos:      LD      @ + 1(1)
0F5C   1E                  RR
0F5D   940B                JP      Even
0F5F   D47F     Odd:       ANI     07F
0F61   01                  XAE
0F62   C280                LD      E(2)
0F64   DC30                ORI     030        ;Segments E & F
0F66   CA80                ST      E(2)
0F68   9007                JMP     Cont
0F6A   01       Even:      XAE
0F6B   C280                LD      E(2)
0F6D   DC06                ORI     06         ;Segments B & C
0F6F   CA80                ST      E(2)
0F71   BA24     Cont:      DLD     Count (2)
0F73   9CE5                JNZ     Npos
                ;Display current position
0F75   C401     Show:      LDI     H(Kybd)
0F77   37                  XPAH    3
0F78   C484                LDI     L(Kybd)-1
0F7A   33                  XPAL    3
0F7B   3F                  XPPC    3
0F7C   902A                JMP     Coma       ;Command key
0F7E   40                  LDE
0F7F   98F4                JZ      Show
0F81   03                  SCL
0F82   FC06                CAI     6          ;1-5 allowed
0F84   94EF                JP      Show
0F86   C40F                LDI     H(Pos)
0F88   35                  XPAH    1
0F89   C418                LDI     L(Pos)
0F8B   02                  CCL
0F8C   70                  ADE
0F8D   31                  XPAL    1
0F8E   C100                LD      (1)
0F90   02                  CCL
0F91   F4FF                ADI     —1
```

```
0F93   02               CCL
0F94   F9FF             CAD    —(1)
0F96   9402             JP     Fine 2     ;Valid move
0F98   90DB             JMP    Show
0F9A   C225   Fine 2:   LD     Key(2)
0F9C   9C03             JNZ    Firstn
0F9E   40               LDE
0F9F   CA25             ST     Key(2)     ;First key press
0FA1   60     Firstn:   XRE               ;Not first press
0FA2   9E43             JNZ    Disp(2)    ;not allowed
0FA4   B900             DLD    (1)        ;Make move
0FA6   9243             JMP    Disp(2)    ;Display result
0FA8   C225   Coma:     LD     Key(2)     ;Mem pressed
0FAA   9A43             JZ     Disp(2)    ;You haven't moved!
0FAC   C403   Go:       LDI    3
0FAE   CA24             ST     Count(2)
0FB0   C40F             LDI    H(Pos)
0FB2   35               XPAH   1
0FB3   C418             LDI    L(Pos)
0FB5   31               XPAL   1
0FB6   C400             LDI    0
0FB8   01               XAE
0FB9   C101   Try:      LD     +1(1)
0FBB   02               CCL
0FBC   FD02             CAD    @+2(1)
0FBE   C904             ST     4(1)
0FC0   60               XRE               ;Keep nim sum
0FC1   01               XAE
0FC2   BA24             DLD    Count(2)
0FC4   9CF3             JNZ    Try
0FC6   40     Solve:    LDE
0FC7   980E             JZ     Nogo       ;Safe position
0FC9   E100             XOR    (1)
0FCB   03               SCL
0FCC   FD02             CAD    @+2(1)
0FCE   94F6             JP     Solve
0FD0   02               CCL
0FD1   F1F9             ADD    —7(1)      ;Make my move
0FD3   C9F9             ST     —7(1)
0FD5   923F             JMP    Ymove(2) ;Now you, good luck!
0FD7   C405   Nogo:     LDI    05
0FD9   CA24             ST     Count(2)   ;Make first move
0FDB   C5FF   No:       LD     @—1(1)
0FDD   02               CCL
0FDE   F4FF             ADI    —1
0FE0   02               CCL
0FE1   F9FF             CAD    —1(1)
0FE3   9406             JP     Fine
0FE5   BA24             DLD    Count(2)
0FE7   9CF2             JNZ    No
0FE9   9307             JMP    +7(3)      ;i.e. Abort—I lose
0FEB   B900   Fine:     DLD    (1)        ;Make my move
0FED   923F             JMP    Ymove(2) ;now you chum.
       0000             .END
```

Music

'Function Generator' produces a periodic waveform by outputting values from memory cyclically to a D/A converter. It uses the 8-bit port B of the RAM I/O chip to interface with the D/A, and Fig. 1 shows the wiring connections. The D/A chosen is the Ferranti ZN425E, a low-cost device with a direct voltage output.

Any waveform can be generated by storing the appropriate values in memory. The example given was calculated as an approximation to a typical musical waveform.

'Music Box' plays tunes stored in memory in coded form. The output can be taken from one of the flag outputs. Each note to be played is encoded as one byte. The lower 5 bits determine the frequency of the note, as follows:

Rest	A	A#	B	C	C#	D	D#	E	F	F#	G	G#
00	01	02	03	04	05	06	07	08	09	0A	0B	0C
	0D	0E	0F	10	11	12	13	14	15	16	17	18

There are two octaves altogether.

The top three bits of the byte give the duration of the note, as follows:

Relative Duration:	1	2	3	4	5	6	7	8
	00	20	40	60	80	A0	C0	E0

Thus for any specific note required the duration parameter and frequency parameter should be added together. A zero byte is reserved to specify the end of the tune.

The program uses two look-up tables, one giving the time-constant for a delay instruction determining the period of each note and the other giving the number of cycles required for the basic note duration.

'Organ' generates a different note for each key of the keyboard by using the key value as the delay parameter in a timing loop. Great skill is needed to produce tunes on this organ.

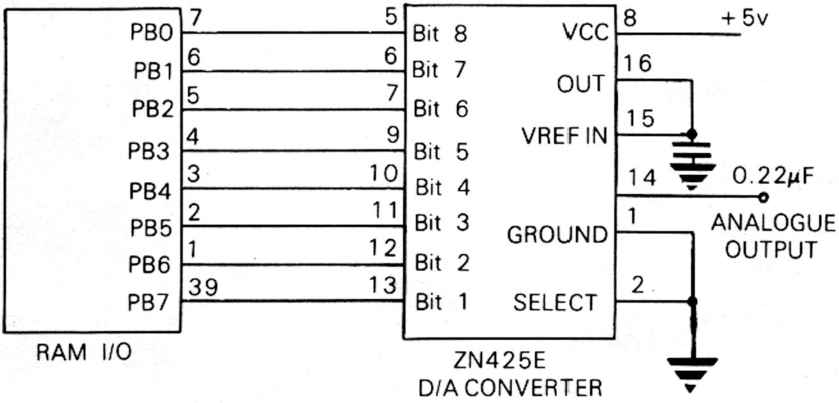


Fig. 1

Function Generator

```
                        ; Generates arbitrary waveform by outputting
                        ; values to D/A Converter.
                        ; uses Ram I/O chip. (Relocatable).
                        ;
                        Portb       =       0E21
                        E           =       -128      ;Extension as offset
                        ;
0000                                .=0E80            ;Start of Ram in Ram/IO
0E80    C40F            Start:      LDI     H(Endw)
0E82    36                          XPAH    2
0E83    C448                        LDI     L(Endw)
0E85    32                          XPAL    2         ;P2-> End of waveform
0E86    C40E                        LDI     H(Portb)
0E88    35                          XPAH    1
0E89    C421                        LDI     L(Portb)
0E8B    31                          XPAL    1
0E8C    C4FF                        LDI     X'FF      ;All bits as outputs
0E8E    C902                        ST      +2(1)     ;Output definition B
0E90    C4D8            Reset:      LDI     -Npts
0E92    02                          CCL
0E93    01              Next:       XAE
0E94    C280                        LD      E(2)      ;Get next value
0E96    C900                        ST      (1)       ;Send to D/A
0E98    40                          LDE
0E99    F401                        ADI     1         ;Point to next value
0E9B    98F3                        JZ      Reset     ;New sweep
0E9D    04                          DINT              ;Equalize paths
0E9E    90F3                        JMP     Next      ;Next point
                        ;
                        ;
                        ; Sample waveform of 40 points
                        ; Fundamental amplitude 1
                        ; 2nd Harmonic amplitude 0.5 zero phase
                        ; 3rd Harmonic amplitude 0.5 90 deg. lag.
                        ;
                        ; Equation is:
                        ; Sin(X)+0.5*Sin(2.0*X)40.5*Sin(3.0*X-0.5*π)
                        ; With appropriate normalization
                        ;
0EA0                                .=0F20
                        ;
0F20                    Wave:       .BYTE   077,092,0B0,0CB,0E1,0ED
0F26                                .BYTE   0EF,0E6,0D5,0BE,0A5,08E
0F2C                                .BYTE   07F,077,076,07D,087,092
0F32                                .BYTE   09B,09E,09A,090,080,06F
0F38                                .BYTE   05C, 04D,042,03D,03D,040
0F3E                                .BYTE   046,04B,04D,04D, 04A,046
0F44                                .BYTE   044,047,050,060
            0F48        Endw        =
            0028        NPTS        =         Endw-wave ;No. of points
            0000                    .END
```

Music Box

```
                ; Plays a tune stored in memory
                ; 1 Byte per note
                ; top 3 bits = duration (00-E0) = 1 to 8 units
                ; bottom 5 bits = note (01-18) = 2 octaves
                ;
0000                     . = 0F12
                ;Table of notes
0F12            Scale:   .BYTE    0            ;Silence
0F13                     .BYTE    0FF,0EC,0DB,0CA,0BB,0AC
0F19                     .BYTE    09E,091,085,079,06E,063
0F1F                     .BYTE    059,050,047,03F,037,030
0F25                     .BYTE    029,022,01C,016,011,00C
                ;Table of cycles per unit time
0F2B                     .BYTE    044,048,04C,051,055,05B
0F31                     .BYTE    060,066,06C,072,079,080
0F37                     .BYTE    088,090,098,0A1,0AB,0B5
0F3D                     .BYTE    0C0,0CB,0D7,0E4,0F2,0FF
                ;Program now:
0F43            Cycles:  . = . + 1
0F44            Count:   . = . + 1
                ;
0F45    3F      Stop:    XPPC     3            ;'Go, 'term', to play again
                ;Begin execution here
0F46    C40F    Begin:   LDI      H(Scale)
0F48    35               XPAH     1
0F49    C40F             LDI      H(Tune)
0F4B    36               XPAH     2
0F4C    C490             LDI      L(Tune)
0F4E    32               XPAL     2            ;P2 points to tune
0F4F    C601    Play:    LD       @+1(2)       ;Get next note code
0F51    01               XAE                   ;Save in ext.
0F52    40               LDE
0F53    98F0             JZ       Stop         ;Zero = terminator
0F55    1C               SR
0F56    1C               SR
0F57    1C               SR
0F58    D4FC             ANI      X'FC
0F5A    C8E9             ST       Count
0F5C    C412             LDI      L(Scale)
0F5E    01               XAE
0F5F    D41F             ANI      X'1F         ;Get note part
0F61    02               CCL
0F62    70               ADE                   ;no carry out
0F63    31               XPAL     1            ;Point P1 to note
0F64    C100             LD       (1)          ;Note
0F66    01               XAE                   ;Put it in ext.
0F67    C118    Hold:    LD       +24(1)       ;Cycle count
0F69    C8D9             ST       Cycles
0F6B    40      Peal:    LDE
```

```
0F6C    9C04                JNZ     Sound    ;Zero = silence
0F6E    8F80                DLY     X'80     ;Unit gap
0F70    9011                JMP     More
0F72    8F00     Sound:     DLY     X'00
0F74    06                  CSA
0F75    E407                XRI     X'07     ;Change flags
0F77    07                  CAS
0F7B    B8CA                DLD     Cycles
0F7A    9807                JZ      More
0F7C    08                  NOP              ;Equalize paths to
0F7D    C410                LDI     X'10     ;Prevent clicks in
0F7F    8F00                DLY     X'00     ;Sustained notes
0F81    90E8                JMP     Peal
0F83    B8C0     More:      DLD     Count
0F85    94E0                JP      Hold
0F87    8F20                DLY     X'20     ;Gap between notes
0F89    90C4                JMP     Play     ;Get next note
                 ;
0F8B                        . = 0F90
0F90             Tune:      .BYTE   02D,02D,02F,04C,00D,02F
0F96                        .BYTE   031,031,032,051,00F,02D,
0F9C                        .BYTE   02F,02D,02C,02D,00D,00F
0FA2                        .BYTE   011,012,034,034,034,054,
0FA8                        .BYTE   012,031,032,032,032,052,
0FAE                        .BYTE   011,02F,031,012,011,00F·
0FB4                        .BYTE   00D,051,012,034,016,032
0FBA                        .BYTE   071,06F,08D,0

        0000                .END
```

Organ

```
                 ; Each key on the keyboard generates a
                 ; different note (though the scale is
                 ; somewhat unconventional!) Relocatable.
                 ;
                            . = 0F1F
0F1F             Count:     . = . + 1
        0D00     Disp:      =       0D00     ;Display & keyboard
                 ;
0F20    C40D     Enter:     LDI     H(Disp)
0F22    35                  XPAH    1
0F23    C400     New:       LDI     L(Disp)
0F25    31                  XPAL    1
0F26    C408                LDI     08
0F28    C8F6                ST      Count    ;Key row
0F2A    C501     Again:     LD      @+1(1)
0F2C    E4FF                XRI     0FF      ;Key pressed?
0F2E    9808                JZ      No
0F30    8F00                DLY     00       ;Delay with AC = key
0F32    06                  CSA
0F33    E407                XRI     07       ;Change flags
```

```
0F35    07                  CAS
0F36    90EB                JMP      New
0F38    B8E6      No:       DLD      Count
0F3A    9CEE                JNZ      Again
0F3C    90E5                JMP      New

        0000                .END
```

Miscellaneous

'Message' gives a moving display of segment arrangements according to the contents of memory locations from 'Text' downwards until an 'end-of-text' character with the top bit set (e.g. 080). Each of the bits 0-6 of the word in memory corresponds, respectively, to the seven display segments a-g; if the bit is set, the display segment will be lit. Most of the letters of the alphabet can be formed from combinations of the seven segments: e.g. 076 corresponds to 'H', 038 to 'L', etc. The speed with which the message moves along the display depends on the counter at OF2D. If the first and last 7 characters are the same, as in the sample message given, the text will appear continuous rather than jumping from the end back to the start.

'Reaction Timer' gives a readout, in milliseconds, of the time taken to respond to an unpredictable event. To reset the timer the 'O' key should be pressed. After a random time a display will flash on. The program then counts in milliseconds until the 'MEM' key is pressed, when the time will be shown on the display.

The execution time of the main loop of the program should be exactly one millisecond, and for different clock rates the delay constants will have to be altered:

Rate	Location:	OF2A	OF37	OF39
1 MHz		07D	0A8	00
2 MHz		OFA	0A1	01
4 MHz		OFF	093	03

'Self-Replicating Program' makes a copy of itself at the next free memory location. Then, after a delay, the copy springs to life, and itself makes a copy. Finally the whole of memory will be filled by copies of the program, and from the time taken to return to the monitor one can estimate the number of generations that lived.

Message

```
                        ; Displays a moving message on the
                        ; 7-segment displays
                        ;
                        ;
                        ;
0000                            .=OF1F
OF1F                    Speed:  .=.+1
                        ;
OF20      C40D          Tape:   LDI     H(Disp)
OF22      35                    XPAH    1
OF23      C400                  LDI     L(Disp)
OF25      31                    XPAL    1
OF26      C40F          Go:     LDI     H(Text)
OF28      36                    XPAH    2
OF29      C4CA                  LDI     L(Text)-8
OF2B      32                    XPAL    2
OF2C      C4EO          Move:   LDI     X'EO      ;Determines sweep speed
```

```
0F2E   C8F0             ST      Speed
0F30   C407   Again:    LDI     7
0F32   01     Loop:     XAE
0F33   C280             LD      -128(2)
0F35   C980             ST      -128(1)
0F37   C4FF             LDI     X'FF
0F39   02               CCL
0F3A   70               ADE                 ;i.e. decrement ext.
0F3B   94F5             JP      Loop
0F3D   B8E1             DLD     Speed
0F3F   9CEF             JNZ     Again
0F41   C6FF             LD      @-1(2)      ;Move letters
0F43   94E7             JP      Move        ;X'80 = end of text
0F45   90DF             JMP     Go
                ;
       0D00     Disp    =       0D00
                ;
                ; A sample message
                ; Message is stored backwards in memory
                ; first character is 'end of text', X'80.
                ; For a continuous message, first and
                ; last seven characters must be the
                ; same (as in this case).
                ;
0F47                    . = 0FA0
0FA0                    .BYTE   080,079,079,06D,040,037
0FA6                    .BYTE   077,039,040,03E,03F,06E
0FAC                    .BYTE   040,06D,077,040,06E,03E
0FB2                    .BYTE   07F,040,079,037,030,071
0FB8                    .BYTE   040,06E,038,038,03F,01F
0FBE                    .BYTE   040,077,040,06D,030,040
0FC4                    .BYTE   039,040,071,03F,040,06D
0FCA                    .BYTE   040,079,079,06D,040,037
0FD0                    .BYTE   077,039
       0FD2     Text    = .                         ;start of message

       0000                .END
```

Self-Replicating Program

```
                ; Makes a copy of itself and then
                ; executes the copy.
                ; Only possible in a processor which permits
                ; one to write relocatable code, like SC/MP
                ;
       FFFC     LDX     =       Head-Loop-1  ;offset for load
       000D     STX     =       Last-Store-1 ;offset for store
                ;
0000                    . = 0F12
0F12   C4FC   Head:     LDI     LDX
0F14   01               XAE
0F15   C080   Loop:     LD      -128(0)      ;PC-relative-ext = offset
```

```
0F17    01                XAE
0F18    02                CCL
0F19    F411              ADI     STX-LDX
0F1B    01                XAE
0F1C    C880    Store:    ST      -128(0)       ;ditto
0F1E    40                LDE
0F1F    03                SCL
0F20    FC10              CAI     STX-LDX-1     ;i.e. increment ext.
0F22    01                XAE
0F23    40                LDE
0F24    E414              XRI     Last-Loop-1   ;finished?
0F26    9CED              JNZ     Loop
0F28    8FFF              DLY     X'FF          ;shows how many copies
0F2A            Last      = .                   ;were executed.
        0000              .END
```

Reaction Timer

```
                ; Gives readout of reaction time in milliseconds
                ; display lights up after a random delay
                ; Press'MEM' as quickly as possible.
                ; Press '0' to play again. (Relocatable)
                ; 150 = excellent, 250 = average, 350 = poor
                ;
        01F4    Cycles    =       500           ;SC/MP cycles per msec
        0F00    Ram       =       0F00
        0D00    Disp      =       0D00
        0005    Adlh      =       5
        000C    Adl       =       12
        000E    Adh       =       14
        015A    Dispa     =       015A          ;'Address to segments'
                ;
0000                      . = 0F20
0F20    C401    Begin:    LDI     H(Dispa)
0F22    37                XPAH    3
0F23    C459              LDI     L(Dispa)—1
0F25    33                XPAL    3
0F26    C205              LD      Adlh(2)       ; 'Random' number
0F28    01      Wait:     XAE
0F29    8F7D              DLY     Cycles/4
0F2B    02                CCL
0F2C    70                ADE                   ;Count down
0F2D    94F9              JP      Wait
0F2F    C903              ST      + 3(1)        ;Light'8' on display
0F31    40                LDE                   ;Now zero
0F32    CA0C              ST      Adl(2)
0F34    CA0E              ST      Adh(2)
                ;Main loop ; length without DLY = 151 µcycles
0F36    C4A8    Time:     LDI     (Cycles-151 — 13)/2
0F38    8F00              DLY     0
0F3A    03                SCL
0F3B    C20C              LD      Adl(2)
```

```
0F3D    68                  DAE
0F3E    CA0C                ST       Adl(2)
0F40    C20E                LD       Adh(2)
0F42    68                  DAE
0F43    CA0E                ST       Adh(2)
0F45    40                  LDE
0F46    02                  CCL
0F47    F903                CAD      +3(1)     ;Test for key
0F49    98EB                JZ       Time
0F4B    3F        Stop:     XPPC     3         ;Go display time
0F4C    90FD                JMP      Stop      ;Illegal return
0F4E    90D0                JMP      Begin     ;Number key
                  ;
0F50                        . = 0FF9           ;Pointers restored
                  ;                            ;From ram
0FF9    0D00                .DBYTE   Disp      ;P1-> Display
0FFB    0F00                .DBYTE   Ram       ;P2-> Ram
        0000                .END
```

MK 14
VDU Instructions

The MK14 VDU is memory mapped and works by DMA (Direct Memory Access) of the MK14 memory. It must be connected to the address bus and the data bus of the MK14. Each time the VDU needs to read the memory (so that it can display its contents),the VDU sends a signal to stop the SCMP (this signal is called NENIN). It then takes NRDS low and counts up twice through the addresses on A0 – A7 meanwhile displaying their contents. The memory page (1 page = 256 bytes) selected depends on the inputs PS1 – PS4 (page selects) which correspond to A8 → A11. If these are changed half way through a vertical scan of the TV two different pages may be displayed. The VDU requires only a +5 volts stabilised power supply from the MK14 regulator. A heatsink will be necessary.

Construction Notes.

1. We recommend you use sockets for all the integrated circuits.
2. When soldering use a minimum of solder and a fine tipped soldering iron.
3. The holes are plated through and you should not solder both sides of the board.
4. Check the board carefully for any flaws.
5. Carefully read all the notes we supply. If you are truly uncertain about how to proceed contact us for more information.
6. Use reasonable caution when handling CMOS components.

Component List.

Part Number	Value	Remarks
R1, R16	470	Yellow Violet Brown
R2, R11, R14, R17	1K2	Brown Red Red
R3, R4	27K	Red Violet Orange
R5, R7, R10, R12, R13, R15	4K7	Yellow Violet Red
R8	2K4	Red Yellow Red
R9	150	Brown Green Brown
R6	–	Not required
C1, C2, C3, C4, C5, C6, C7		Any value between 30N and 100N
C8	~~6N8~~ 10n	~~6N8 or 682~~ 0·01
C9	220	220K or 221
C10	–	Not required
C11	~~82~~ 100	~~821~~ 101
D1		Blue body. White band + ve.
Q1, Q2, Q3, Q4	BC239	
IC1	74L86	
!C2	74LS20	May be 74L20
IC3	74LS93	
IC4	74LS74	May be 74L74
IC5	4011	May be 5611
IC6	4040	May be 5640
IC7, IC12	74LS04	May be 74L04
IC8	74LS157	May be 81L22 or 74L157
IC9, IC10	80L95	
IC11	74LS27	
IC13	4012	May be 5612
IC14	74LS00	May be 74L00
IC15	DM8678CAB	Character generator
IC16	74LS165	May be 8590

Astec UM1233 Modulator
Printed Circuit Board.

First of all make the following connections. If you have an issue 4 or issue 5 board the connections can be made through a double sided connector at the top of the MK14 board.

VDU connection*	Name	MK14 connection		Remarks
a1	0V	pin 20 of	IC1	Zero volts
a2	A0	pin 25	"	Address bus
a3	A1	pin 26	"	"
a4	A2	pin 27	"	"
a5	A3	pin 28	"	"
a6	A4	pin 29	"	"
a7	A5	pin 30	"	"
a8	A6	pin 31	"	"
a9	A7	pin 32	"	"
a10	A8	pin 33	"	"
a11	A9	pin 34	"	"
a12	A10	pin 35	"	"
a13	A11	pin 36	"	"
a18	D0	pin 16	"	Data bus
a19	D1	pin 15	"	"
a20	D2	pin 14	"	"
a21	D3	pin 13	"	"
a22	D4	pin 12	"	
a23	D5	pin 11	"	"
a24	D6	pin 10	"	"
a25	D7	pin 9	"	"
a28	NRDS	pin 2	"	Negative read strobe
a31	NENIN	pin 3	"	Stop processor
a32	+5V	pin 40	"	Power supply

*looking at the component side of the PCB, row a is closest to the end of the board and connections 32 are at the side nearest to Q3 and Q4.

VDU connection	Name	MK14 connection	Remarks
b9	PS1	Hard wired to +5V, 0V, or an IO port	These determine which pages are displayed by the VDU. As a first test connect PS1, PS2, PS3, PS4 to 0V.
b10	PS2		
b11	PS3		
b12	PS4		
b13	VDU OFF	FLAG 1	Take low (natural state on reset) to turn VDU on.
b14	GRAPHICS/CHARS	Can connect to flag or be switched	Take low for character mode, high for graphics.
b15	REVERSE PAGES	"	Take low to reverse top and bottom pages.
b16	INVERT VIDEO	"	Take low to give reverse video (black on white).
b17	TOP PAGE	–	High when first half of TV picture displayed.
b27	XOUT	pin 38 of IC1	Clock signed at 4Mhz. (not 4.43 Mhz).

It is also necessary to make the following modifications to the MK14 board.

1. Connect a 4K7 resistor from NWDS (pin 1 of IC1) to +5V (pin 40 of IC1).
2. Cut the connection from pin 3 of IC1 (NENIN) to 0V.
3. Cut the connection from pin 3 of IC1 to pin 15 of IC10.
4. Remake connection from pin 15 of IC10 to 0V.

When first testing the VDU connect b9 – b12 to 0V and leave b14 – b17 unconnected. On powering up the MK14 the LED display should work normally. If the output of the UHF modulator is connected to the aerial socket of a UHF TV you should be able to find a strong signal from the VDU on channel 36 showing a bit-map of the monitor.

You will notice that the top half and lower half of the screen are the same. Now disconnect b11 from 0V and connect it to b17. The screen will now display the first two ¼K pages of memory (the monitor PROM). If you connect b16 to 0V the picture will invert. If you connect b14 to 0V the display will show the contents of memory as ASCII characters. If you connect b15 to ground the top and bottom pages will swap round.

Next try connecting b9, b11 and b12 through a 1K resistor to +5V. The VDU will now display the normal RAM (0F00 – 0FFF) and extra RAM (0B00 → 0BFF) on the screen. Notice how some locations flicker as they are continuously changed by the program in the monitor PROM. This flicker disappears if the RESET button is depressed. Try connecting b14 to b17 to produce a display which is part graphics and part characters. Notice that you can change the characters by writing new values into the extra RAM area (20_{16} corresponds to a space). You can change the pattern in the graphics area by writing into 0F12 → 0FFF. (00_{16} corresponds to a blank). Notice how the eight bits of each byte are spread out in a row.

Finally write 02 into 0FFF and press GO. (This is a quick way of setting FLAG 1). The VDU display should blank and the MK14 will run at full speed so that you can load taped programs. Press reset to turn on the VDU again.

NB: When running the VDU causes program execution to run about 6% slower.

The following three programs illustrate how to use an MK14 fitted with the extra RAM, RAM–I/O chip and VDU. If the Extra RAM and normal RAM are displayed by the VDU the only RAM area remaining for user program is the RAM of the RAM-I/O chip. The following three programs fit into these 80_{16} bytes. One can of course use part of the VDU RAM for larger user programs but then they will appear on the display. It is straightforward to add more RAM if this proves necessary.

BIT is a subroutine that can be used to turn on and off and read the spots of a graphics display. It forms the basis of a graphics program. (b14 should be taken to +5V through a 1K resistor).

PUTC is a subroutine which makes the MK14 VDU behave like a standard VDU. It puts the ASCII character corresponding to the byte in the accumulator (lower six bits only) onto the screen in consecutive locations and handles the codes for Carriage Return, Line Feed, Vertical Tab, Backspace and Horizontal Tab. (b14 should be wired to 0V).

SHOWCH is a short demonstration program which clears the screen and then displays the font of the character generator. (b14 should be wired to 0V).

A Short Technical Description.

The MK14 VDU uses a 74LS74, a 74LS93 and a 4040 as a counter chain to count the 312 lines of a TV display and to generate row and column addresses for character or graphics display.

A 74LS157 is used to select the different mappings required for graphics bit map (eight bytes in a row) and for character display (16 bytes in a row). Two 80L95s isolate the VDU from the address and data bus when it is being used by the SCMP. A 4011 generates the sync pulse waveforms. A 74L86 is used to invert the video and buffer the CMOS. Further buffering is performed by the two 74LS04s.

A 74LS165 parallel in/serial out shift register generates the graphics by shifting out the data read on the databus and a DM8678 character generator chip is used to generate ASCII characters from the lower six bits of the databus.

The other chips control the use of the address and databus by the SĊMP and VDU and provide the necessary signals for the shift register and character generator.

Two transistors are used to buffer NENIN, XOUT and a further two to form a composite video signal which is fed into the onboard modulator.

BIT PROGRAM FOR MK14 VDU

```
0880    02                BIT     CCL             P1 should point to the page to be displayed.
0881    CAFF                      ST BIT(2)       This routine requires P2 to point to a stack.
0883    C201                      LD Y(2)         0(2) should contain X and 1(2) contain Y
0885    IE                        RR              where 0<X<63 and 0<Y<31.  Values outside
0886    IE                        RR              these ranges are mapped on modulo 64 and 32.
0887    IE                        RR              If on entry the accumulator contains
0888    IE                        RR              a) 00 the bit at (X,Y) is cleared.
0889    IE                        RR              b) 01 the bit at (X,Y) is set
088A    D4F8                      ANI X'F8        c) FF the bit at (X,Y) is read and the value
088C    01                        XAE             returned in the accumulator (0 for zero).
088D    C200                      LD X(2)
088F    1C                        SR
0890    1C                        SR
0891    1C                        SR
0892    D407                      ANI X'07
0894    70                        ADE
0895    31                        XPAL 1          X and Y are used to calculate PIL.
0896    C200                      LD X (2)
0898    D407                      ANI X'07
089A    02                        CCL
089B    F424                      ADI X'24        Use PC relative extension register addressing
089D    01                        XAE             to obtain a suitable mask for the bit
089E    C080                      LD E(0)         corresponding to (X, Y).
08A0    CAFE                      ST MASK (2)
08A2    C2FF                      LD BIT (2)
08A4    940A                      JP PUT
08A6    C100              GET:    LD 0(1)         Read bit
08A8    D2FE                      AND MASK (2)
08AA    9814                      JZ RET
08AC    C401                      LDI X'01
08AE    9010                      JMP RET
08Bo    9802              PUT:    JZ S            If zero clear a bit.
08B2    C2FE                      LD MASK (2)     Set a bit
08B4    01                S:      XAE
08B5    C2FE                      LD MASK (2)
08B7    E4FF                      XRI X'FF
08B9    D500                      AND 0(1)
08BB    70                        ADE
08BC    C900                      ST 0(1)
08BE    C2FF                      LD BIT (2)
08C0    3F                RET:    XPPC 3
08C1    90BD                      JMP BIT
08C3    8040201008040201                          Table of mask values
```

SHOW CHARS ON MK14 VDU

```
0880    C40F    SHOWCH    LDI X'0F
0882    35                XPAH 1
0883    C400    S1        LDI X'00
0885    31      S2        XPAL 1
0886    C420              LDI X'20
0888    CD01              ST @ 1(1)
088A    31                XPAL 1
088B    9CF8              JNZ S2
088D    35                XPAH 1
088E    E40C              XRI X'0C
0890    9804              JZ S3
0892    C40B              LDI X'0B
0894    90EC              JMP S1
0896    31      S3        XPAL 1
0897    C40B              LDI X'0B
0899    35                XPAH 1
089A    C443              LDI X'43
089C    C902              ST 2(1)
089E    C448              LDI  X'48
08P0    C903              ST 3(1)
08A2    C441              LDI X'41
08A4    C904              ST 4(1)
08A6    C452              LDI X'52
08A8    C905              ST 5(1)
08AA    C43D              LDI X'3D
08AC    C906              ST 6(1)
08AE    A807    S4        ILD 7(0)
08B0    C907              ST 7 (1)
08B2    8FFF              DLY X'FF
08B4    90F8              JMP S4
```

This routine requires only the RAM-I/0 chip and Extra RAM to be fitted. It first blanks the screen and then displays the character set.

PUTC ROUTINE FOR MK14

```
0880    01      PUTC    XAE                 The character whose ASCII code is in
0881    02              CCL                 the accumulator is written to the next
0882    C40B            LDI H(EXTRA RAM)    character cell on the screen.
0884    35              XPAH 1              Carriage Return, Line Feed, Vertical Tab,
0885    40              LDE                 Backspace and Horizontal Tab are interpreted.
0886    E40D            XRI 00D
0888    981B            JZ CR
088A    E407            XRI 007
088C    981C            JZ LF
088E    E401            XRI 001
0890    981D            JZ VT
0892    E403            XRI 003
0894    9828            JZ BS
0896    E401            XRI 001
0898    9803            JZ HT
089A    40              LDE
089B    C900            ST 0(1)
089D    31      HT      XPAL 1
089E    F401            ADI 001
08A0    31      S2      XPAL 1
08A1    40              LDE
08A2    3F              XPPC 3
08A3    90DB            JMP PUTC
08A5    31      CR      XPAL 1
08A6    D4F0            ANI 0F0
08A8    90F6            JMP S2
08AA    31      LF      XPAL 1
08AB    F410            ADI 010
08AD    90F1            JMP S2
08AF    C400    VT      LDI 000
08B1    CAFF            ST–1(2)
08B3    31      S1      XPAL 1
08B4    C420            LD! 020
08B6    C900            ST 0(1)
08B8    AAFF            ILD–1(2)
08BA    9CF7            JNZ S1
08BC    90E2            JMP S2
08BE    31      BS      XPAL 1
08BF    F4FF            ADI 0FF
08C1    90DD            JMP S2
```

Science of Cambridge Limited

6 King's Parade
Cambridge CB2 1SN
Telephone Cambridge (0223) 311488