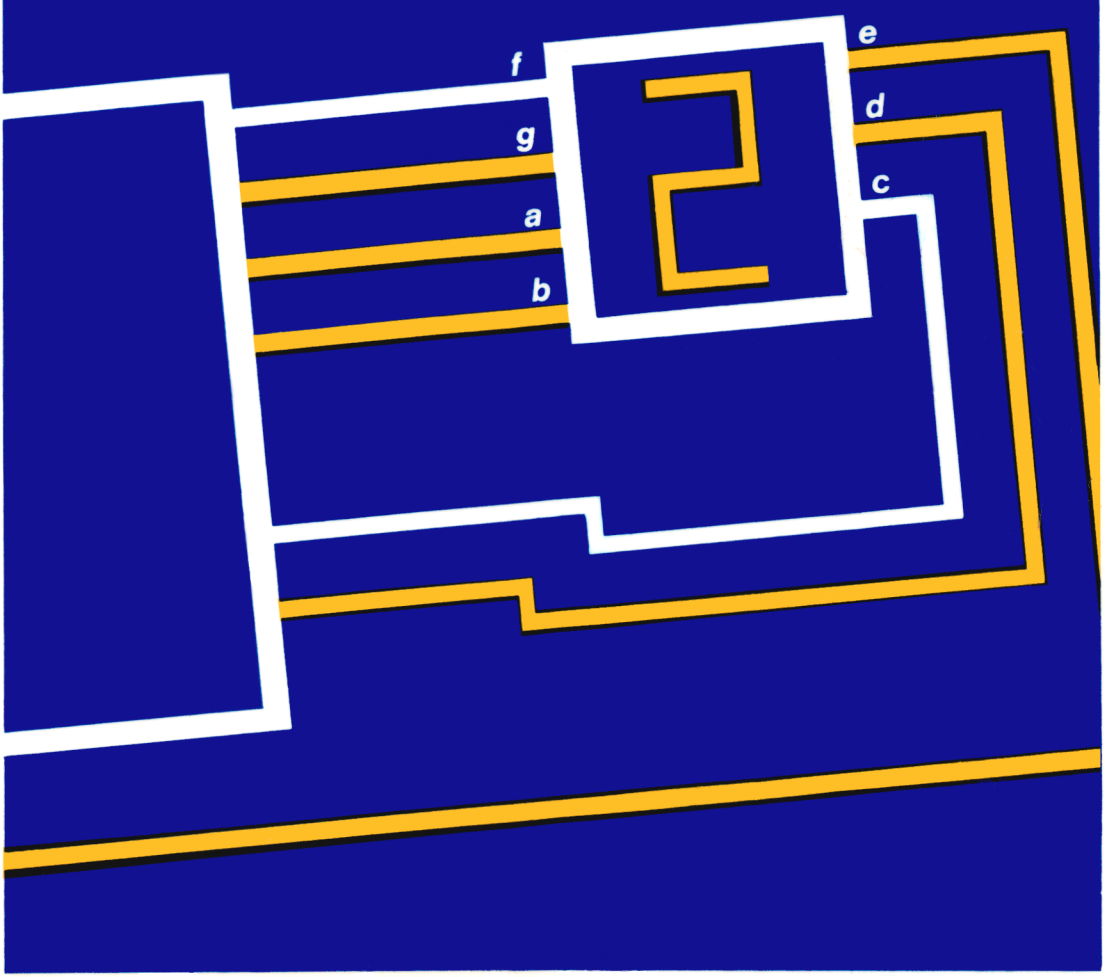
MEgaCyCAL Software limited

electronics series

DIGITAL DEVICES



First published by SciCAL Software Ltd. 1983.

DIGITAL DEVICES

Program and text
by
G. B. CLAYTON

THE DIGITAL DEVICES PACKAGE

The program and booklet which make up this package have been designed to provide a modelling and investigation approach for exploring some of the digital devices on which complex electronic digital equipment is based. Models of more basic electronic devices including logic gates, flip flops, half and full adders, and a binary ripple counter are covered in the SciCAL package "D-LOGIC". If you are new to digital electronics you are advised to start with "D-LOGIC" before progressing to this package.

The program in this package models the following devices:

— a four bit Binary/B.C.D. up/down counter with parallel load and clear facility

- a seven segment display, and binary to seven segment display decoder, in which the user draws up the decoder truth table

- a counter, decoder and seven segment display system

- an 8-bit shift register with facilities for parallel loading or serial load shift left or shift right and recirculate facility

- a ROM model

- a RAM model

The program enables you to experiment with the models following the experiments suggested in this booklet or by creating your own experiments. Interaction with the models gives you a thorough understanding of the functional behaviour of these devices, and gives you access to knowledge of a wider range of systems than can be covered realistically in practical work.

How to load the program

BBC Model B : 40 and 80 track disc systems

Hold down the SHIFT key and tap the BREAK key.

BBC Model B : Cassette systems

1. Type CHAIN " " and press RETURN.

2. Place the cassette in your tape recorder and press the PLAY button. When the program has loaded, if your tape recorder does not have motor control, switch off the recorder when the program tells you to do so.

3. Do not rewind the tape. Because of the length of this program several loads are involved. Each section chains the next so that when you have finished with one section you should follow the program's instructions for loading the next section.

Running the program

The program should be used in conjunction with this booklet which gives explanations about the devices, tips for making the best use of the program's versatility, and full instructions for experiments. Some sections of the program run from three computer keys only, and operating instructions are given in the relevant sections of the booklet.

DIGITAL DEVICES

Introduction.

This package explores the action of some of the digital devices which act as the functional building blocks for making complex electronic digital equipment. Practical digital devices are in the form of complete circuits which are formed on a silicon chip. These are called integrated circuits (I.C.s). Advances in I.C. technology continue to allow the development of devices of increasing complexity (larger scales of integration) and account for the comparatively low cost of such digital instruments as pocket calculators and digital watches. A single large scale integrated circuit (L.S.I.) may contain all the electronic circuitry needed for a complete system or instrument in a single package.

I.C. chips are encapsulated in small circuit packages, and external electrical connections to the chip are made by way of pins which are attached to the package. The effective use of I.C. devices does not demand a detailed knowledge of their internal circuitry. What is needed is an understanding of the relationships which exist between the electrical signals at the external pin connections of the device.

Experimentation with practical devices can be a valuable aid to understanding but it is very time consuming because of the many electrical connections which have to be made to the pins on the device. This CAL package saves you time by allowing you to experiment with models which simulate the behaviour of real devices.

In order to achieve maximum benefit from the use of the package it is suggested that you work through it a section at a time by reading about the device in this booklet first and then running the appropriate program. The book gives you general information about a device and its applications, the program enables you to gain added insight and understanding as a result of your experimentation with the models.

General features of digital devices and the computer models.

The electrical signals which are applied to the input pins of digital devices and those which appear at their output pins are all two state signals. They may be either a high voltage level or a low voltage level. Actual voltage levels are not of significance, they must simply be different in order to distinguish clearly between two possible states.

The computer programs produce graphic models of devices in which input and output connections to a device are represented by lines. The state of a line, its logic state High or Low, is represented in the model by the colour of the line or by the symbols "1" or "0" shown next to the line. The program user changes the logic state of input lines by pressing the appropriate keys on the keyboard. Output lines respond to changes in input in accordance with the nature of the specific device being examined.

The state of a line which exists at any instant provides a single unit of binary data which is called a Bit. A group of bits is called a digital word, and word lengths are frequently in multiples of eight bits. An eight bit word is called a Byte.

There are two different ways in which binary data (digital words) are transmitted from one part of a system to another, and these are called parallel and serial transmission. In parallel transmission all bits in a word occur simultaneously on a group of lines, each bit has its own line. In serial transmission bits are transmitted in time sequence on a single line, all bits in a word exist for equal time periods on this single line.

Digital devices perform a variety of different operations on binary data. Significance is attached to an operation by the assignment of a code to the data. Once a code is assigned to data, manipulation of data becomes in effect manipulation of coded information.

Many different codes are used but they are all binary codes in the sense that they are based on only two characters, the digits "1" and "0". There are numerical codes in which digital words represent numerical magnitude and character codes represent characters. Characters include letters of the alphabet, digits and symbols such as punctuation marks.

In following sections of this booklet you will be introduced to commonly used binary codes. You will become familiar with the techniques used in the manipulation of binarily coded information through your interaction with the program models.

SECTION 1: COUNTERS

A counter is a digital device which is used to count and give an indication of the number of binary pulses which are applied to it in some period of time. By a binary pulse we mean a logic level transition sequence Hi to Lo to Hi, or Lo to Hi to Lo.

Counting is a functional operation which is common to the working of many different digital electronic systems. For example, a digital frequency meter measures the frequency of a repetitive signal by counting the number of cycles of the signal which occur in a fixed time interval. A digital watch measures time by continuously counting pulses generated by a stable frequency oscillator. Your computer uses a counter called the program counter which keeps track of the computer's progress as it works its way through a program.

Numerical codes.

In order to understand the operation of a counter we must first examine ways of representing numerical information. We are all so familiar with counting that we tend to overlook the logical ideas underlying any counting process.

All number systems are essentially codes which assign a one to one correspondence between a set of symbols (together with their associated sounds) and a set of discrete object or event quantities.

In the decimal number system there are ten basic symbols, the numerals 0,1,2,3,4, 5,6,7,8,9, and any decimal number can be represented by a combination of these basic symbols. In a multi-digit number each symbol carries weight according to its position in the number. For example the number 369 is understood as:

Hundreds	Tens	Units
3	6	9

$$369 = 3 \times 10^2 + 6 \times 10^1 + 9 \times 10^0$$

The numerical codes used in digital systems employ only the two symbols "1" and "0". Multi-digit numbers are made up as a combination of these two symbols. There are several binary numerical codes in common use, the most frequently used is called natural binary.

The Natural Binary Numerical Code.

The natural binary code is similar to the decimal code in that it is a positional weighted code. Each digit (each bit) in a natural binary code word is weighted according to its position in the word. Whereas in the decimal system weightings are in powers of ten, in the natural binary system weightings are in powers of two.

For example the natural binary number 1011 is understood as:

Positional weights

2^3 2^2 2^1 2^0

1 0 1 1

The decimal equivalent of the natural binary number

1011 is $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 11$ (dec).

The first bit in the number, the one at the left hand side, carries the biggest weighting and is called the most significant bit (MSB). The bit at the right hand side carries the least weighting and is called the least significant bit (LSB). Shifting the position of a bit one place to the left increases its value by a factor of two, shifting one place to the right decreases its value by a factor of two.

Binary Coded Decimal.

Most people do not readily appreciate the significance of numerical information displayed as a sequence of "1"s and "0"s. Digital instruments like digital voltmeters and digital watches display numerical information in decimal form. They use a binary code in which each decimal digit is separately coded in binary form. A minimum of four bits are needed to code the ten symbols of the decimal system. The most frequently used code is called Binary Coded Decimal, B.C.D.

The B.C.D. code uses the first ten states of a four bit natural binary counting sequence to represent in order the ten characters of the decimal system. The six upper states of the four bit code are not used in B.C.D.

The representation of a multi-digit decimal number in B.C.D. requires the use of four bits for each decimal digit. B.C.D. codes thus occur as multiples of four bits. For example the decimal number 729 is represented in B.C.D. as:

7 2 9

0111 0010 1001

Hexadecimal Code.

A number system called Hexadecimal (HEX) is sometimes used as a means of expressing the binary code word stored in a digital register or counter. HEX is not a binary code itself, it is a positional weighted code which uses the sixteen basic symbols 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F, to represent the decimal numbers 0 to 15.

The positional weights carried by the symbols in a HEX number are in powers of sixteen. For example the HEX number A5F is to be understood as:

Positional weights

$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
A	5	F
(dec. eq. 10)	(dec. eq. 5)	(dec. eq. 15)

The decimal equivalent of the HEX number A5F is

$10 \times 256 + 5 \times 16 + 15 \times 1 = 2655$

A unique HEX symbol exists for all sixteen states of a four bit binary word. HEX notation is particularly useful in microcomputers where digital word lengths are normally in multiples of four. HEX provides a concise notation for expressing the contents of digital storage registers. Only two HEX characters are required to distinguish between the 256 states of a byte (an eight bit word).

The Counter Model.

The program provides a model which simulates the behaviour of a four bit counter, and a choice between a natural binary count and a B.C.D. count is available. The functional behaviour of the model is very similar to that of two practical I.C. counters, the types SN74193(Binary) and SN74192(B.C.D.) devices. The main features of the model are illustrated by the screen print out reproduced in Fig. 1.

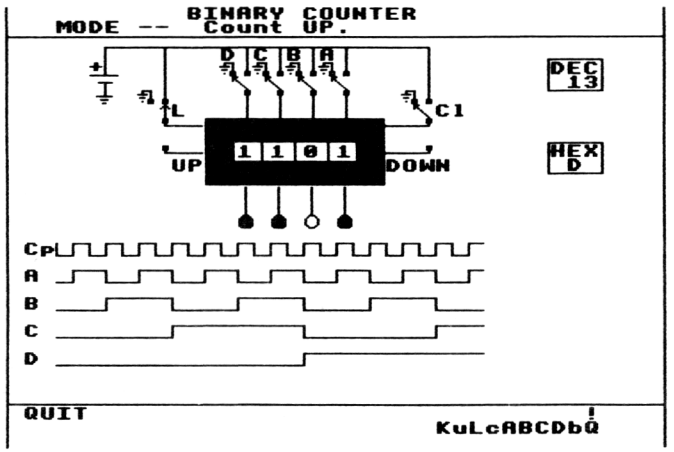


Fig. 1. Screen print out of a four bit counter.

A facility for presetting the state of the counter prior to the counting operation is provided by the four data input lines shown connected to the upper edge of the model and labelled A,B,C,D. A is the LSB, D is the MSB. If the control line labelled L is brought Lo the output lines change to agree with the input lines independently of the count pulses.

Presetting a counter with data present on a set of input lines is sometimes referred to as a parallel loading of the counter. In this particular counter model the parallel loading operation is called an asynchronous parallel load. The term asynchronous refers to the independence of the loading operation on pulses applied to the count input.

There are some counter devices which incorporate synchronous parallel loading. In a synchronously loaded counter, loading takes place in synchronism with the next pulse applied to the count input after the load input has been set to its "load" state. Synchronous parallel loading is incorporated in other program models in this package. A control line labelled C1 when a Hi is applied to it forces all bits to "0". The effect of this clear input is independent of the load and count inputs and over-rides them.

There are two inputs to which pulses being counted can be applied, a count up input and a count down input. Pulses are applied to one while the other is at its Hi state. Pulses applied to the count up input cause the count to increment up through a binary sequence, while pulses applied to the count down input cause the count to decrement down through this sequence.

User interaction with the model.

In order to experiment with the model you need use only three of the keys on your computer keyboard, namely the cursor shift left and shift right control keys and the RETURN key. A list of letters signifying various commands is displayed at the lower right hand edge of your display screen. The cursor shift keys are used to position a pointer which moves over this list of command letters. The function performed by a particular command is indicated at the lower left hand edge of the screen. Pressing the RETURN key causes the indicated command to be executed.

A reminder of the type of counter model (binary or B.C.D.) and the mode of operation in which it is currently working is displayed in the top two lines of the screen. Note that counting is inhibited if either the Clear line is Hi or the Load line is Lo. In the models a line in its Hi state ("1") is shown in red.

Here are some procedures for performing two simple experiments with the model.

Experiment 1: Incrementing the counter.

Procedure:

1. Set the clear line C1 to Lo. The logic state of the clear line is changed by positioning the command pointer over the command letter "c" and pressing RETURN.
2. Set the load control line L to Hi (command letter "L" and RETURN).
3. Apply clock pulses to the count up input. UP/DOWN inputs are selected by the command letter "u" (RETURN). Clock pulses are applied by selection of command letter "K" (RETURN).

As the count proceeds you will see an indication of the count provided by the states of the output lines shown along the lower edge of the model. Binary, decimal and HEX representations of the count are displayed on the screen. Voltage waveforms which develop at outputs A,B,C,D, are also displayed.

Repeat the procedure with the clock pulses applied to the count down input.

Repeat again with the model changed to perform a B.C.D. counting sequence (Command letter "b" RETURN).

Experiment 2: Parallel loading the counter.

Procedure:

1. Set the states of the parallel load inputs to the values you require to load the counter with a specific number (command letters A,B,C,D, followed by RETURNs).

2. Set the load input L to Lo. The states of the input lines will be loaded into the counter register and will appear at the output lines (command letter L then RETURN).

SECTION 2: A SEVEN SEGMENT DISPLAY

Many electronic instruments, such as electronic calculators and digital watches, are required to provide a visual display of numerical information. A variety of indicator devices have been developed to perform the display function. All indicator devices employ some form of element which emits light in response to an electrical signal.

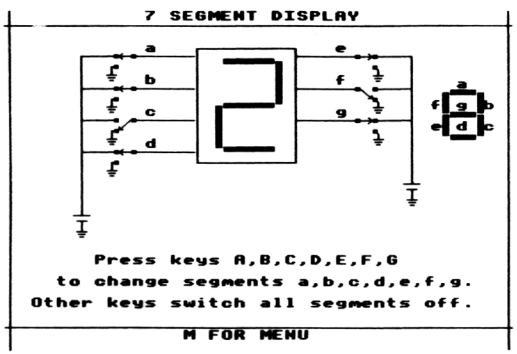


Fig. 2. Print out of a Seven Segment Display Model.

The most frequently used numerical indicator devices employ a seven segment structure with either liquid crystal elements (Liquid Crystal Display, L.C.D.) or light emitting diode elements, L.E.D.s. The format of the segments in a seven segment display is shown by the screen print out in Fig. 2. The segments labelled a,b,c,d,e,f,g, each have an input control line, and the logic signal applied to the control line determines whether or not a particular segment is illuminated.

Any of the decimal digits 0-9 can be displayed by excitation of the appropriate light emitting segments, for example when segments a,c,d,e,f,g, are illuminated the decimal number 6 is displayed. As you run the program you will soon find which segments must be excited in order to display a particular decimal number.

SECTION 3: DECODERS (CODE CONVERTERS)

A digital word conveys information in coded form. Quite frequently it is necessary to detect the presence of specific code words and change them into some other

form of code. A decoder or code converter is a device which is used to perform this function. A decoder is a combinational logic circuit which accepts a parallel binary word at its input and produces a binary output which indicates the presence or absence of specific input words.

The functional operation which is performed by a decoder can best be illustrated by a truth table which relates all its possible input and output states. You will appreciate the significance of such a truth table when you run the program in this section. The screen display produced in this section is illustrated in Fig. 3.

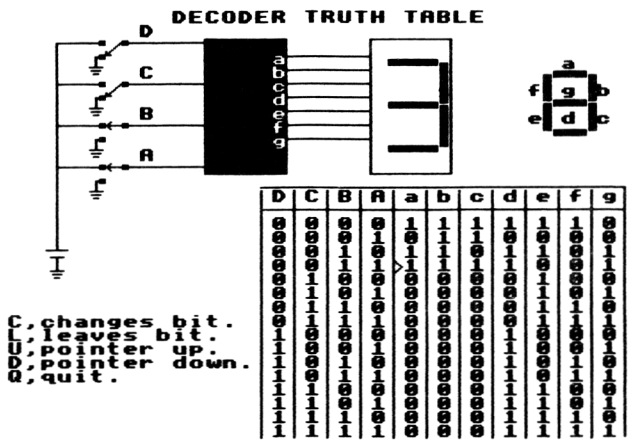


Fig. 3. Screen print out of a Decoder Truth Table.

A decoder having four input lines is simulated. The sixteen possible states of the input lines are shown in the four left hand columns of the displayed truth table. You are required to enter the values of the output codes which you wish the decoder to generate for each state of its input code.

The output lines of the decoder are connected as inputs to the seven segment display model. You enter the output codes which make the display produce a required character. For example, in the first row of the truth table the input code is 0000 corresponding to the decimal character 0. The seven segment display produces the character 0 when segments a,b,c,d,e,f, are illuminated and segment g is extinguished, so you enter a "1" in the output columns a,b,c,d,e,f, and an "0" in column g.

In this program you are in effect specifying the decode function which you want the decoder to perform. On quitting the program you are sent straight into the next

program in which the decoder you have created has its inputs supplied by the output lines of a counter model. If you choose a B.C.D. counter only the first ten rows of the decoder truth table will be involved. If you use a binary counter all sixteen states of the decoder truth table will be incremented. In this case you will have to enter codes in the six upper states of the decoder truth table which will cause the display to generate the alphabetical characters of a HEX code. The capital letters B and D are not possible, you will have to make do with lower case b and d.

SECTION 4: COUNTER, DECODER AND SEVEN SEGMENT DISPLAY.

This program section, illustrated by the screen print out in Fig. 4., lets you see how the functional elements of a counter system, counter, decoder and display work together. If you enter the program directly from the menu, the decoder will have its decode function correctly set up. If you enter the program from the previous section the decoder will have the decode function which you yourself specified in the previous section.

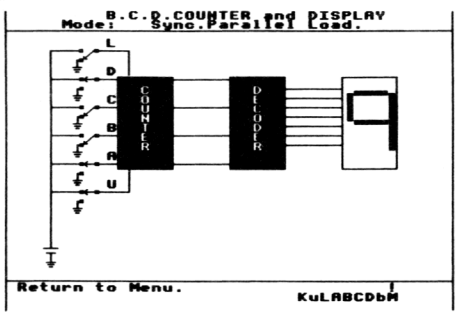


Fig. 4. Screen print out of a Counter, Decoder and Display.

You interact with the model through the counter command system as in Section 1. If you find the display is not producing the characters you require go back to the previous section of the program and correct the decoder truth table.

SECTION 5: SHIFT REGISTERS.

Devices which are used to store single bits of binary data are called flip-flops. The action of a single flip-flop is examined in another CAL package in this series, "D-LOGIC". A combination of flip-flops arranged so as to store several bits of binary data is called a digital register. A digital register in which the stored bits

can be moved or shifted from one storage element to the next adjacent element is called a digital shift register. The program gives you a model of an eight bit shift register to experiment with. You are advised to run the program concurrently with your reading of this present section of the text.

The Eight Bit Shift Register Model.

The main features of the shift register model are illustrated by the screen print out given in Fig. 5. The model simulates in many respects the behaviour of the M.S.I. digital integrated circuit device type SN54198.

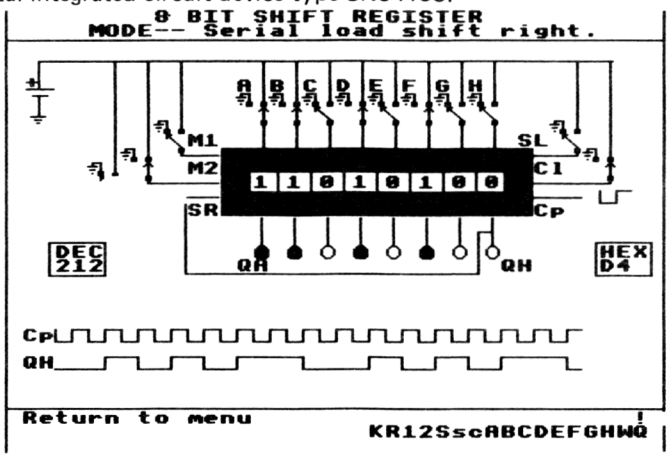


Fig. 5. Screen print out of an Eight Bit Shift Register.

The bit values stored in the eight storage elements of the register are indicated by the logic states of the eight output lines, and associated L.E.D.s shown connected along the lower edge of the model.

The eight lines shown along the top edge of the model are data input lines. The data present on these lines can be parallel loaded into the register. There are two serial input lines labelled SR and SL. Data present on the SR input can be loaded in serial form first into the bit A position and then shifted right into the register. Data present on the SL input can be loaded first into the bit H position and then shifted left into the register.

Parallel loading, shifting left and shifting right are operations which take place in synchronism with pulses applied to the Cp input. These pulses are called clock pulses. Transitions take place in synchronism with the leading edge of a clock pulse, that is the Lo to Hi transition of the pulse.

The mode of operation is determined by the logic levels applied to two control inputs labelled M1 and M2. M1 Lo and M2 Hi gives the serial load shift right mode. M1 Hi and M2 Lo gives the serial load shift left mode. M1 and M2 both Hi gives the parallel loading mode, M1 and M2 both Lo is an inhibit mode, which means that the register does not respond to clock pulses when in this mode. There is an asynchronous over-riding clear input control line labelled C1. If C1 is made Lo all bits in the register are set to "0" and no changes can be made to the register contents while C1 remains Lo.

User Interaction with the Shift Register Model.

Interaction with the model is by a series of commands. The command system is similar to that in Section 1, and involves the use of only three keys, the cursor shift left and shift right keys and the RETURN key. Pressing the RETURN key causes execution of the command displayed in the lower left hand corner of the screen. A reminder of the mode of operation in which the model is working is displayed in the top part of the screen. There are seventeen possible commands so it might take you a little time to become familiar with all of them. Once you have had a little practice you will find that the three key command structure gives you a rapid and convenient way of interacting with the model. Here are some suggestions for experiments you can perform which illustrate practical applications of shift registers.

Serial to Parallel and Parallel to Serial Conversions.

An eight bit word can be shifted left or shifted right into the register a bit at a time. It is then available in parallel form at the register outputs. Alternatively a word can be parallel loaded into the register by applying it to the data inputs of the register. The register is then put in a shift mode, eight clock pulses are applied and the word appears in serial form at the output line at the end of the register.

Experiment 1: To serial load the register with 10011010.

Procedure:

1. Observe state of C1 line and make sure it is Hi. To change its state, position the command pointer over the command letter "c" and press RETURN.

2. Use the command letter "W" to select a display of the time sequence (the waveform) of the bit values in the bit A position, displayed as QA.

3. Set M1 Lo, M2 Hi, serial load shift right mode.

4. Make SR Lo corresponding to the LSB in the digital word to be serial loaded. Apply a clock pulse.

5. Make SR Hi corresponding to the next bit and apply another clock pulse. Continue in this way until all eight bits have been loaded. The serial word will now be available as an eight bit parallel word at the data outputs of the register.

You should note that in the displayed waveform a time axis is assumed with time proceeding in the direction left to right in the display. The bit level shown at the left of the waveform corresponds to the first bit applied. This bit occupies the right most position in the register when the serial loading operation has been completed. We assign the right most position in the register as the LSB position, and with this assignation the first bit occuring in the serial word is to be interpreted as the LSB. Decimal and HEX numbers which are displayed to indicate the numerical contents of the register at any time treat the right most register position, (H), as the LSB and the left most position, (A), as the MSB.

Experiment 2: To parallel load the register with 10101101.

Procedure:

1. Make sure C1 is Hi.

2. Set the states of the input lines A,B,C,D,E,F,G,H to correspond with the bit values to be loaded with A the MSB, H the LSB.

3. Make M1 Hi and M2 Hi. (The parallel load mode).

4. Apply a clock pulse. The word should now be stored in the register.

5. Use the command "W" to select display of the QH waveform.

6. Make M1 Lo and M2 Hi. (The shift right mode).

7. Apply clock pulses. The word which you parallel loaded will come out in serial form at QH.

A Shift Register used to perform Scaling Operations.

A shift register can be used to multiply or divide a binary number in factors of two. Shifting the binary data stored in a register one place to the left has the effect of multiplying the natural binary number represented by that data by a factor of

two. Shifting the data one place to the right has the effect of division by two.

Experiment 1: Using the register to perform multiplication.

Procedure:

1. Parallel load the register with the natural binary equivalent of the decimal number 6, (00000110). See the previous experiment for parallel loading.

2. Make the SL line Lo.

3. Make M1 Hi and M2 Lo. (The shift left mode).

4. Apply a clock pulse and note the effect that this has on the number stored in the register.

5. Apply a second and then a third clock pulse. Each time you apply a pulse you will see that the stored number is multiplied by two.

The important thing to remember about this multiplication process is that the register must be long enough to accommodate the largest number expected in the multiplication. If it is not long enough bits will be lost and the stored number will be in error. Also note that binary "0"s are shifted into the right most position as the data is moved to the left.

Experiment 2: Using the shift register to perform division.

Procedure:

1. Parallel load the register with the binary equivalent of the decimal number 198 (11010000).

2. Set the SR input Lo.

3. Set M1 Lo and M2 Hi. (The shift right mode).

4. Apply a clock pulse and note the effect on the number stored in the register.

5. Apply further clock pulses. As each pulse is applied you will see that the number stored in the register is divided by two.

You should note that in this division process the binary point has been assumed to be immediately to the right of the LSB in the register. Any non zero bits shifted out to the right are lost. The effect of this is that the number in the register is the integer part of the number obtained as a result of division by two.

A recirculating Shift Register.

If a shift register is to store data yet at the same time allow the data to be read out in serial form, the data must be fed back into the register at the left as it is read out from the right. This operation is referred to as recirculation.

A recirculating shift register when continuously clocked generates a repetitive binary sequence corresponding to the data stored in the register. Many logic systems require a sequence of precisely spaced timing pulses for initiating a series of operations. A recirculating shift register can be used to provide such a sequence. Output pulses from the bit registers are connected to the logic circuits whose sequence is to be controlled.

Experiment 1: Making the register recirculate.

Procedure:

1. Parallel load the register with some specific word (say 10110100) using the procedure previously described.

2. Connect the recirculate line to the model using command letter "R" and RETURN.

3. Select the serial load shift right mode, M1 Lo, M2 Hi.

4. Apply a series of clock pulses command letter "K" and RETURN.

Examine the waveform generated at QA or QH. You will see a fixed pattern or sequence which repeats for every eight clock pulses.

Experiment 2: Recirculating Register used as a Ring Counter.

Procedure:

1. Parallel load the register with a Hi in one bit and all the rest Lo.

2. Connect the recirculate line to the model.

3. Select the serial load shift right mode.

4. Apply a series of clock pulses.

The system acts as a divide by eight ring counter. A single output pulse is produced for every eight clock pulses.

SEMICONDUCTOR MEMORIES.

The ability to store or remember information is fundamental to the operation of many digital systems. Stored information may consist of numbers to be used in computation, or numbers obtained as a result of computations, or it may be non numerical information like instructions to be used to direct a system operation. Whatever the nature of the information it is always stored in binary coded form as a sequence of bits.

The temporary storage of a single digital word is performed by a digital register. You should be familiar with the operation of a storage register as a result of your interaction with the shift register model of the previous section. The bit storage capacity of a register is of course limited by the length of the register.

Semiconductor devices capable of storing many bits of data are called semiconductor memories. Two types of memory device called read only memory (ROM), and random access memory (RAM) are dealt with in this package, which concentrates on external operating principles rather than the internal circuit principles underlying the operation of the memories.

ROMs and RAMs are functionally very similar types of device. Both allow the storage of many digital words with the ability to gain rapid access to any stored word. The terminology used to distinguish the two devices is a little confusing in that both types of device in fact provide random access to stored words. If stored words are thought of as entries in a table of values, random access means that it is not necessary to go through the table reading all the values in order to find a particular entry in the table.

There are important differences between ROMs and RAMs. The binary data stored in a ROM is permanently stored there at the time of its manufacture and cannot be changed as a part of normal device operation. Data stored in a ROM is not lost when the device has power switched off.

Data stored in a RAM is temporarily stored, and it can be changed as a normal part of the device's operation with new data "written" into the RAM. Data stored in a RAM is lost when the power is switched off.

As you run the ROM and RAM programs, your interaction with the models will make clear the significance of the terms "read" and "write" and will enable you to understand the differences between the two types of memory device.

SECTION 6: THE ROM MODEL.

Section 6 of the program gives you a ROM model illustrated by the screen print out in Fig. 6. It is usual to characterise the bit storage capacity of a ROM in terms of the number of words stored in the memory and the number of bits in each word. The ROM shown in Fig. 6. designated as an 8 x 4 ROM stores eight words of four bits each.

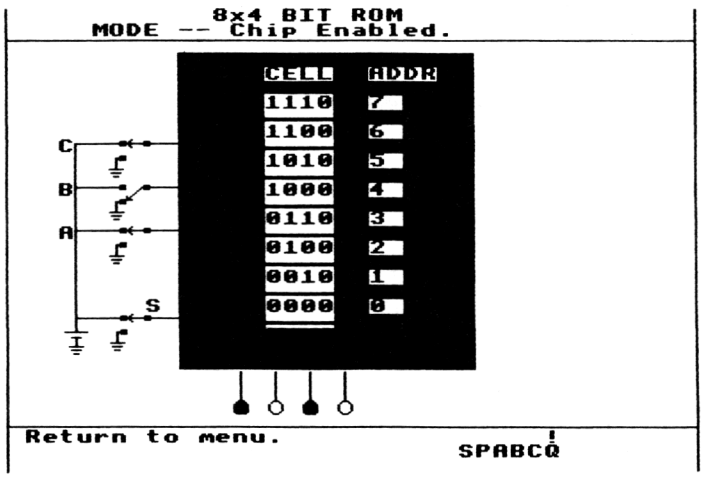


Fig. 6. Screen print out of a ROM Model.

The physical nature of the storage elements in a ROM need not concern us. From the operational point of view it is convenient to regard data as simply stored at locations or cells in memory. Eight such locations are represented in Fig. 6. as rows of four bit words. The locations are numbered 0,1,2,3,4,5,6,7. This numbering serves to distinguish between the different locations, and is called the address of the memory cell.

Four input lines are shown in Fig. 6. attached to the left hand edge of the model. Lines A,B,C constitute the so called address bus ("bus" is a term used to describe a group of lines). The logic levels which are applied to the address lines select a particular memory location and cause the contents of that location to appear at the data output lines which are shown at the lower edge of the model.

The input line labelled S is the "chip select" input. If S is made Lo the memory is disabled and the output lines take on a high impedance state which is neither

Hi nor Lo. This facility allows several devices to share a common data bus. When a device is disabled, its data output lines are electrically isolated from the data bus although still physically connected to it. Only the device which is required to put data on the bus is enabled at any one time.

The word storage capacity of a ROM determines the number of input address lines required by the ROM. The word length, i.e. the number of bits in a word, determines the number of data output lines. You can change the storage capacity of the model ROM with the command letter "N".

The data to be stored in a ROM must be specified at the time of its manufacture and cannot then be changed. The model allows you to simulate this initial data specification. Selection of the command letter "P" allows you to change the data stored in the model ROM.

A ROM performs what is known as a table look up function. Memory locations can be considered as entries in a large table of values. By applying an address code to the ROM we are in effect looking up one particular entry in this table. Data and instructions which are being used continuously by your computer are stored in a ROM. For example, a ROM is used to store the set of instructions for the BASIC language you use to communicate with your computer.

A ROM can be used as a code converter. The input code is made equal to the ROM's address code, so that the desired output code can be found in the memory location specified by the input or address code.

A ROM can be used to perform mathematical operations. A very simple operation is illustrated by the memory contents shown in Fig. 6., namely multiplication by two. The output code is the input address code multiplied by two. Much more complicated operations can easily be performed. The logarithm, the sine or indeed any function of the address code could just as easily be programmed into the memory locations.

SECTION 7: THE RAM MODEL.

The main features of this model are illustrated in Fig. 7. A 64 x 8 bit RAM is simulated, 64 x 8 signifying a storage of 64 words each of 8 bits. A 6 bit address bus is needed to select between the 64 word storage locations ($2^6 = 64$). The address bus is shown connected to the left hand side of the model. Two 8 bit data buses are shown connected to the right hand side of the model. One of these is used for writing data into the memory, the other for reading data out of the memory.

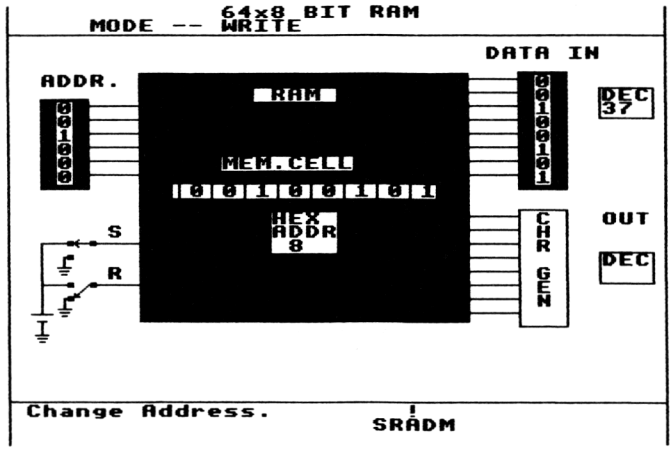


Fig. 7. Screen print out of a RAM Model.

Two control lines labelled S and R are shown. S is the chip select line, and S must be set high to enable operation of the RAM. When S is in its Lo state, read/write operations are prevented and the output lines take on a high impedance state which is neither Hi nor Lo. The state of line R selects between a read or write mode of operation. R Hi gives the READ mode, R Lo gives the WRITE mode.

It is obviously not possible to display the contents of all 64 storage locations on the screen at the same time. The contents of the particular location selected by the address code together with the HEX value of this address code is displayed in the centre of the model.

A system of commands operated in a similar way to that used in the other models allows you to interact with the RAM. You are urged to experiment with the model and find out for yourself the functions performed by the various logic lines. Some suggested experiments appear below.

It is important to note that some practical RAM devices differ from the model in that they have a single data bus which is used for both writing to memory and reading out from memory. RAMs used in microcomputer systems are usually of this type. All the functional elements in a microcomputer system normally share a common data bus which is used for the transfer of data between them. A RAM in its WRITE mode accepts data from the bus and stores it. A RAM in its READ mode puts data into the bus for transfer to another part of the system. A RAM when disabled by the logic state of its chip select input is effectively isolated from the bus.

Experiment 1: Reading data from the RAM.

Procedure:

1. Check that the model is in the READ mode with R Hi.

2. Set the chip select S Hi, using command letter "S".

3. Increment the address code, command letter A (RETURN).

While you manipulate the model you should observe carefully the changing states of the various logic lines. As you increment up the address code you will see the data stored at each address appear on the output lines. Codes corresponding to ASCII characters (decimal 32 to 126) cause those characters to be printed out at the bottom of the display.

On your first entry into the model you will find ASCII character codes stored at address locations 00 to 15 (HEX). Incrementing the address code from 00 to 15 causes a message to be printed out.

N.B. ASCII stands for American Standard Code for Information Interchange. It is the alphanumeric (alphabetic and numeric) code most often used in microcomputer systems. You will find a table of ASCII codes in the user guide to your micro.

Experiment 2: Writing to the RAM.

It is suggested that you overwrite the existing memory contents with the ASCII codes which will generate a message of your own (your name for example).

Procedure:

1. Put the RAM in its WRITE mode (R Lo, S Hi).

2. Start with the address code at 00.

3. Change the data in the data input register until you have the ASCII code for the first letter of your message.

4. Change the address code to 01 and put the ASCII code for the second letter of your message into the input register.

5. Repeat the procedure until you have stored the codes for all the letters in your message.

6. Read back the message you have stored using the procedure outlined in the previous experiment.

EXERCISES

Here are some exercises to test your recall and understanding of the topics covered in this package. If you find difficulty in answering, go back to the appropriate program section for help. Correct answers are given at the end but don't cheat, try all the questions before checking the answers.

1. In the natural binary counter model in Section 1, data inputs A and C are Hi, B and D are Lo. Input L is made Lo and then switched back to Hi. What binary number will be stored in the counter after
   (a) Four pulses are applied to the count UP input?
   (b) Four pulses are applied to the count DOWN input?
   (c) Eleven pulses are applied to the count UP input?
   (d) Eleven pulses are applied to the count DOWN input?

2. Repeat exercise 1 for the B.C.D. counter model in Section 1.

3. The four bit natural binary counter on Section 1 has a repetitive pulse of frequency 100KHz applied to its UP input. What are the frequencies of the repetitive voltage waveforms which you may expect to observe at outputs A,B,C,D.

4. A four bit B.C.D. counter is storing the decimal number 3. Sketch the voltage waveforms which you expect to observe at its output lines if ten count UP pulses are applied to it.

5. What is the effect of making the CL line Lo in the model in Section 1?

6. Use the decoder model in Section 4 to set up a B.C.D. to 7 segment decoder truth table which causes the display to be in error by two (i.e. lagging behind by two).

7. An eight bit shift register is storing the HEX number B9 (left hand MSB). Bit values "1", "0", "1" and "1" are shifted right in sequence into the register. What number is now stored in the register? Give your answer in (a) natural binary, (b) HEX, (c) decimal.

8. The eight bit shift register model in Section 5 is parallel loaded with the natural binary number 00010101. What decimal number is stored in the register after (a) two (b) five "0"s are shifted left into the register?

9. The HEX number CA is stored in the 8 bit shift register model in Section 5. A recirculate line is connected between QH and the shift right input SR. What

number is stored in the register after (a) one (b) four (c) eight shift right operations? Express your answers in natural binary, HEX and decimal. Sketch the voltage waveform which you expect to observe at QH.

10. A ROM with a total storage capacity of 512 bits is organised to store 8 bit words. How many lines must the ROM have in its address bus?

11. Program the ROM model in Section 6 to make it function as a 2 bit multiplier. Hint: Let address lines AB define one 2 bit number while lines CD define the other 2 bit number. Store the product of the two numbers at the address formed by the two numbers.

ANSWERS

1. (a) 1001 (b) 0001
   (c) 0000 (d) 1010

2. (a) 1001 (b) 0001
   (c) 0110 (d) 0100

3. 50 KHz, 25KHz, 12.5KHz, 6.25KHz.

5. All bits reset to "0".

7. (a) 11011011 (b) DB (c) 219

8. (a) 84 (b) 160

9. (a) 01100101, 65, 101.

   (b) 10101100, AC, 172.

   (c) 11001010, CA, 202.

10. 6.

ISBN 0 947956 07 7
Published by Megacycal Software, P.O. Box 6, Birkenhead, Merseyside L43 6XH.
Printed by Instaprint, 1 Hamilton Street, Birkenhead, Merseyside L41 6DJ.