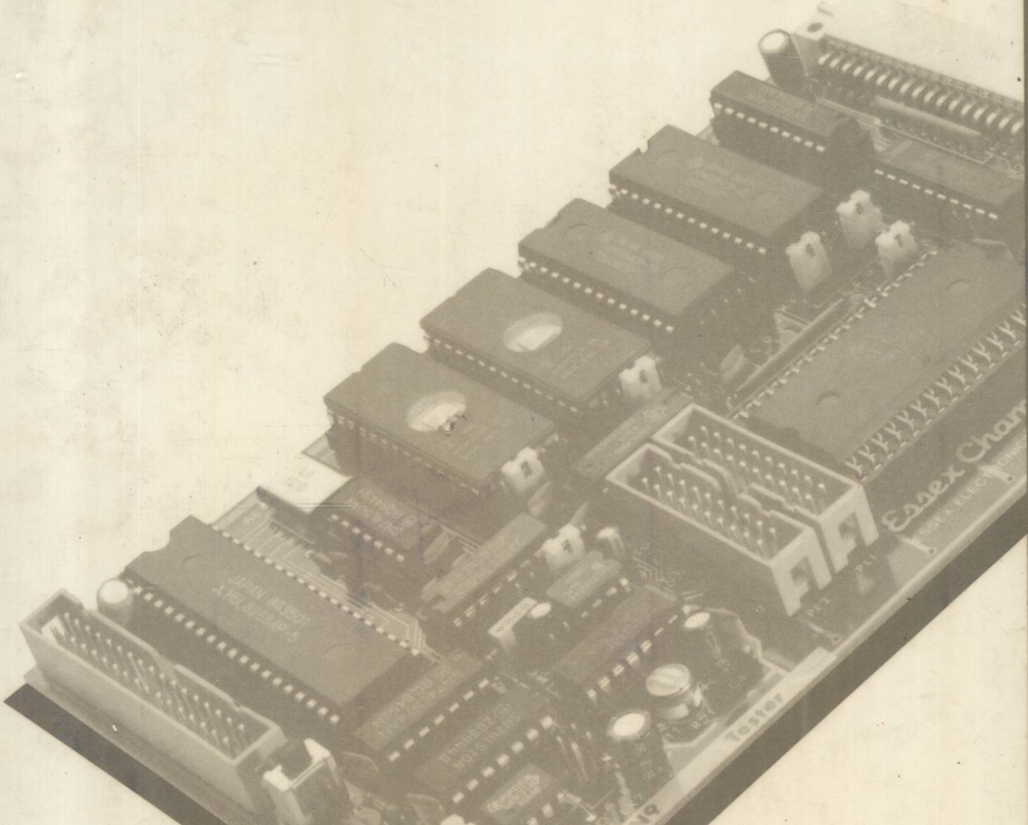
Working with Microprocessors



Ian R. Sinclair

TN47
S616

WORKING WITH MICROPROCESSORS

Ian Sinclair

COLLINS
8 Grafton Street, London W1

Collins Professional and Technical Books
William Collins Sons & Co. Ltd
8 Grafton Street, London W1X 3LA

First published in Great Britain by
Collins Professional and Technical Books 1986

Distributed in the United States of America
by Sheridan House, Inc.

British Library Cataloguing in Publication Data
Sinclair, Ian R.
Working with microprocessors.
1. Microprocessors
I. Title
621.391'6 TK7895.M5

ISBN 0 00 383320 8

Printed and bound in Great Britain by
Mackays of Chatham, Kent

WORKING WITH MICROPROCESSORS

Preface

There are by now very few aspects of machine control that do not feature the use of microprocessors, and the small computer has made its mark in both home and business to an extent that could not have been predicted five years ago. This enormous expansion of the use of microprocessors has not, however, been matched by the availability of *suitable* information. The manufacturers provide a great variety of data books and applications sheets, but these are aimed at the professional designer, who knows what to look for and uses the information more for reference purposes than for learning about microprocessors generally. There must be many design engineers who have been accustomed to analogue circuits, may have had some encounters with TTL digital circuits, but who are now required to work with microprocessor circuits. Similarly, service engineers who may never have seen the circuitry of a microprocessor device are now being required to service increasing numbers of machine-controllers and small computers. In addition, there are many more technically interested readers who would like to know more about the hardware of microprocessor circuits, but have no suitable source of information.

The aim of this book is to deal with the hardware of modern microprocessor devices at a level suitable for the beginner. I don't mean the complete beginner to electronics, because explanations starting from scratch would make the book much too long. I have assumed that the reader knows something of digital systems, in particular the use of binary code and the actions of gates and flip-flops. This is the minimum background knowledge that will be needed for this book, and it seems a reasonable assumption. The principles of gates and flip-flops are summarised in the first chapter, so as to lead to explanation of the operation of the microprocessor. I have not assumed any software knowledge, because this book is concerned predominantly with hardware, but some software knowledge will be an advantage, almost an essential, for any reader who pursues the topics further, particularly to reading the manufacturers' data manuals.

In addition, some topics have been simplified, and the more complex details of others omitted. Modern microprocessor systems are not necessarily complicated systems, but some of the chips that are used, particularly ports, are complicated in the sense that they have to be programmed, and the programming allows such a vast range of options that a full treatment of the chip requires a large manual to itself. Because this book is concerned mainly with hardware, the software programming of ports, for example, has been omitted, along with some of the more exotic options that are available. The aim of this book is to show how microprocessor systems are connected up, what signals exist, the timing relationships, and the actions. These are the topics that the hardware designer and the service engineer particularly need to understand. One complete chapter has been devoted to the problems of servicing microprocessor equipment, and the instruments that are available. Systems have been treated fairly generally, but because examples are always helpful, I have taken examples from a number of devices that are widely used. Of these, the Z80 family is hardly modern though it is very extensively used, but the Intel 8086 and 80256, along with the Motorola 68000 series, are comparatively recent and will be found on a large amount of equipment in years to come. The Texas TMS series has also been mentioned as an example of microprocessor chips that are found mainly in machine-control equipment (programmable controllers).

It is my hope that this book will fill a need that has existed for some considerable time, and that it will be useful for the hard-pressed engineer who has been plunged very suddenly into working with microprocessor equipment. I am very grateful to many firms who provided information, in particular Motorola Ltd, SGS-ATES, Rapid Recall (for Intel data), Texas Instruments, Commodore, GSC (U.K.) Ltd, Thurlby Electronics Ltd, and to Mr P. Mutton of the Essex Electronics Centre, University of Essex. I am also grateful to past students who by their questions made me aware of what was required in a book of this kind. Finally, I am most grateful to the team at Collins, headed by Bernard Watson, who saw the need for this book and commissioned it.

Ian Sinclair

Contents

CHAPTER 1

What is a Microprocessor?

A microprocessor is a programmable logic chip which can make use of memory. Memory in this sense means storage for data which can be written (data stored) or read (data copied). The microprocessor can address this memory, meaning that it can select any part of the memory to store data or to copy existing stored data and make use of it. Within the microprocessor chip itself, logic actions such as the standard NOT, AND, OR and XOR actions can be carried out, as well as a range of other actions such as shift and rotate, and some simple arithmetic. The fact that any sequence of such actions can be carried out under the control of a program is the final item that completes the definition of a microprocessor.

In general, microprocessors are designed so as to fall into one of two classes. One type is intended almost exclusively for industrial control, and this also extends to the control of domestic equipment such as central heating systems. A microprocessor of this type will often be almost completely self-contained, with its own memory built in, and very often this will include the programming instructions. Such microprocessors will very often need to work with a limited number of binary digits (bits) at a time, perhaps 4. The number of possible programming instructions need only be small. The control microprocessor will also be offered typically as a 'semi-custom' device, with the programming instructions put in at the time of manufacture for one particular customer. By contrast, the alternative is the type of microprocessor whose main purpose is computing. The computer type of microprocessor contains little or no memory of its own, but is capable of addressing large amounts of external memory. It will deal with at least 8 bits, and more usually 16 or 32 bits, of data at a time. It has a much larger range of instructions, and will generally operate at higher speeds.

In this book, we shall be concerned with both types of microprocessors, particularly with the many features that they have in common. Before we begin though, it will be useful to look over the history of this remarkable device whose effect on us in this century is comparable with the effect that the steam engine had in the nineteenth century. It's worth recalling, incidentally, that the steam engine was thought to be a device that would cause mass unemployment, yet within a few years of its general adoption as the motive power for factories, reformers were beginning to be concerned about the effects of overemployment, particularly child labour.

The Microprocessor History

The development of the microprocessor was a set of events which consisted partly of accidents, partly of strokes of genius, and partly of good marketing. The origin of the microprocessor was a military contract, placed in the later 1960s, for a programmable controller chip. The contract, like so many of its kind, was cancelled just at the time when production was starting, and the company was left with a production line which had been paid for, but which could make only devices that no-one wanted. The device was what we would now call a 4-bit microprocessor, the Intel 4004. For some time, this looked like the answer to a problem that no one had, but some good marketing activity stimulated engineers to consider the possibilities of a single-chip device which could carry out the actions that until then had required a large assembly of boards. Machine control was one obvious outlet, and the startling new one was the microcomputer. Up to that time, computers depended on constructing central processing units which were very bulky and which depended on fairly small-scale integration at best, often on discrete transistor circuitry. The possibility of a complete computer processing unit in a single-chip form had been thought of, but its emergence as a practical working chip nevertheless took engineers by surprise. For that reason, many of the most significant steps in the use of the microprocessor in the early days appear to have been made by students of electronics, or amateurs experimenting with 'surplus' components.

The 4004 then spawned the 8008, the first 8-bit microprocessor, and this brought about the possibility of really powerful low-cost small-size computers. The Motorola 6800 demonstrated different thoughts about how a microprocessor chip should be designed, and MOSTEK rethought the 6800 and came up with the 6502. This was the microprocessor which Steve Jobs and Steve Wozniak built into their first prototype Apple computer, and which was later used in most of the very successful small computers, including the BBC machine. When, around the same time, the 8008 was developed into the 8080, and Gary Kildall developed the CP/M operating system for small computers, the microprocessor industry as we know it now began the remarkable growth that slowed only in 1985. It's a classic U.S. story of brilliant design, accidents and taking advantage of opportunities. It's rather difficult to imagine anything similar happening in the U.K.

Gates

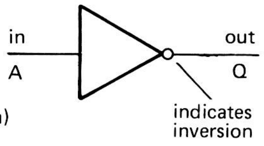
A gate circuit, in electronics, is one which has a number of inputs and (usually) one output. Each input can be driven to one of two states, called 0 or 1. For most purposes, 0 means a voltage level of about 0 to 0.8 V, and 1 means a voltage level between 3.0 V and 5.2 V. The *precise* voltages are unimportant, so long as the two voltages cannot overlap. For any gate circuit, there will be some combination of inputs which will make the output voltage rise to the 1 level. There may be more than one combination of inputs which gives this output, but for all gate circuits it is the *combination* of inputs that is important. For this reason, gate circuits are sometimes called *combinational* circuits. One particularly important feature of combinational circuits is that any circuit action can be obtained by connecting a few basic types of gate circuits. These basic gate types are known as NOT, AND, OR and XOR, and the principles of analysing a logic action into combinations of these gates were first worked out by the Lincolnshire mathemetician George Boole in the early nineteenth century. The method that he devised, called Boolean algebra, is used to this day to analyse devices that he could not possibly have dreamed of.

As far as a microprocessor is concerned, however, all gates are of a fairly simple type, with two inputs and one output. The basic types are the same, but the method by which gates are used is very different. In a hardware circuit, for example, it is possible to act on groups of signals in parallel. You might have eight input signals being applied to four different gate inputs at the same time. The gates in the microprocessor can be used only one at a time, in sequence. This makes our use of gates quite different from the methods that we use in hardware circuits, and if you are used to hard-wired gate circuits, then you need to adjust your thinking considerably. The best place to start is with a reminder of how the basic gate circuits work for the usual one or two inputs that are always used within microprocessors.

The NOT gate operates on a single-bit input. As always with gate circuits, the best way of describing the action is by the use of a truth table, in which each possible input and output is listed. For the NOT gate, the truth table is the very simple one in Figure 1.1. The action of the NOT gate is inversion, giving an output which is the inverse of the input, the other possible logical value. The AND gate obeys the rules of the truth table of Figure 1.2. Two inputs are required for this *logic comparison*, and the output is at level 1 only when both inputs are simultaneously at this level. The other two gate actions also require two inputs each, and their truth tables are illustrated in Figures 1.3 and 1.4 respectively. The OR action makes the output bit equal to level 1 when either input is at level 1, or when both inputs are at level 1. The XOR gate differs slightly from the OR inasmuch as it excludes the case when both inputs are at level 1, hence the name. One way of looking at the two-input XOR gate is that the output is at logic 1 only when the inputs are different (one at 0, the other at 1).

Within the microprocessor, these logic gates are arranged in sets. A microprocessor deals with groups of bits at a time. The earliest

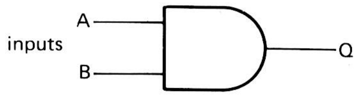
(a)



(b)

A in	Q out
0	1
1	0

Fig. 1.1 The NOT gate or inverter. (a) The symbol which is used in a circuit diagram. Note that the small circle denotes inversion of a signal. If the circle is placed at the input, it means that an inverted signal is needed at the input to operate a device. (b) The truth table for the NOT gate.

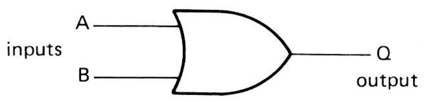


(a)

(b)

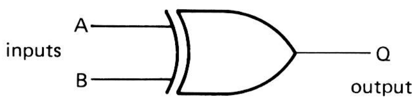
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 1.2 The AND gate, showing (a) the circuit symbol and (b) the truth table. The internal AND gates of a microprocessor are of this type, with two inputs per gate. The international symbol has been used because the alternative BS symbol is seen only in some U.K. examination papers.



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 1.3 The international circuit symbol and truth table for the OR gate. The action is that any input or combination of inputs at logic 1 will cause the output to be at logic 1.



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 1.4 The international circuit symbol and truth table for the XOR gate. The action is almost identical to that of the OR gate, but excluding the case when more than one input is at logic 1.

microprocessors dealt with 4-bit groups, and several industrial controller microprocessors, along with specialised calculator chips, still use 4-bit groups, sometime known as *nibbles.* By far the most common grouping, even in the latter part of the 1980s, is the *byte*, a set of 8 bits, and this grouping is used by both industrial control and computing microprocessors. The later types of computing microprocessors, however, use the 16-bit grouping which is called a *word* or sometimes a *gulp.*

Suppose we consider for a moment the nibble group for a controller microprocessor. This would need to be able to work with two sets of 4-bit groups, and would apply them to the inputs of gates arranged in fours. Each gate would work with the corresponding bits in each group. These bits are conventionally numbered from one end, counting the least-significant bit as bit 0, the next as bit 1 and so on. If the idea of 'significance' is new to you, please look at Appendix A now. The AND gate, for example, in a 4-bit microprocessor, would AND bit 0 of one group with bit 0 of the other group and at the same time AND bit 1 of the first group with bit 1 of the second, and so on. In other words, the logic gate actions are carried out with a set of identical gates in parallel, each gate working on one place of bit. Figure 1.5 illustrates this with the AND action on two 4-bit nibbles. For a 4-bit microprocessor, then, the gating for each action consists of one gate for each bit in the group that is used. An 8-bit microprocessor will need eight gates of each type, and the 16-bit microprocessor will need sixteen gates of each type.

To illustrate how gate actions are typically used, consider a 4-bit processor which is being used to control the action of a washing machine. At one point in the cycle, the controller must switch on the wash motor if (a) there is water at the correct level, (b) the water temperature is correct, (c) the safety switch is closed and (d) the motor is not already moving. Suppose that each of these conditions is represented by 1 bit of a 4-bit nibble. When each condition is true, the corresponding bit is put to logic level 1, so that for *all* conditions true the nibble is 1111. Now if this is made one input of the AND gate, and the other input is also 1111, then the output will also be 1111 but *only* while all 4 bits from the water, temperature, safety-switch and motor sensors are all correct. The program for this microprocessor will therefore carry out the AND action, and test for the result being 1111, upon which the motor will be switched on.

Consider now a very different action, this time using an 8-bit byte

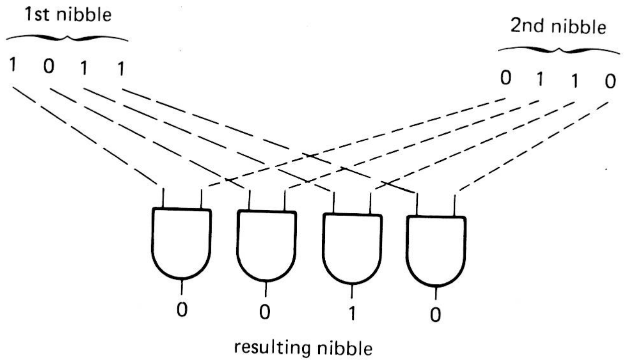


Fig. 1.5 The microprocessor makes use of gates which each have two inputs. Each input is taken from 1 bit in each of two groups (nibble, byte or word). The bits are in corresponding positions in the group, and this illustration shows 2 nibbles being ANDed by four AND gates.

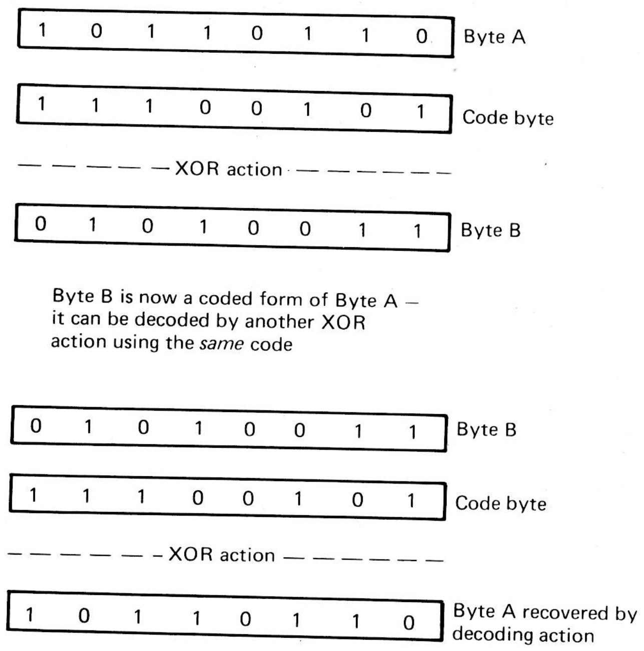

1 0 1 1 0 1 1 0 Byte A
1 1 1 0 0 1 0 1 Code byte
XOR action
0 1 0 1 0 0 1 1 Byte B
Byte B is now a coded form of Byte A – it can be decoded by another XOR action using the *same* code
0 1 0 1 0 0 1 1 Byte B
1 1 1 0 0 1 0 1 Code byte
XOR action
1 0 1 1 0 1 1 0 Byte A recovered by decoding action

Fig. 1.6 Illustrating the coding and decoding action of the XOR gate set when applied to a byte. Using a code of several bytes repeated in sequence can give coded text which is very difficult to decode without knowledge of the code bytes that were originally used.

example, Figure 1.6. A byte is used as a code for a letter of the alphabet. By using the XOR action with another byte, the result is a coded version of the original. This can, in turn, be decoded by another XOR action *with the same code byte*, giving the original byte. This is a method that can be used to code messages with a very good degree of security, provided that a group of code letters are used in sequence.

These examples show something of the range of uses for the gate actions, which are the backbone of all process control actions and are also very important in computing actions.

Registers

A register is a circuit, made up from flip-flops, which can store a set of bits. The flip-flop, remember, is a unit which can have its output set (to logic 1) or reset (to logic 0) by a pulse at its input. An 8-bit register will be able to store 8 bits, a 16-bit register will store 16 bits, and so on. Registers, unlike gates, are clocked devices. This means that their actions are controlled by a pulse, the clock pulse, which is applied to a separate

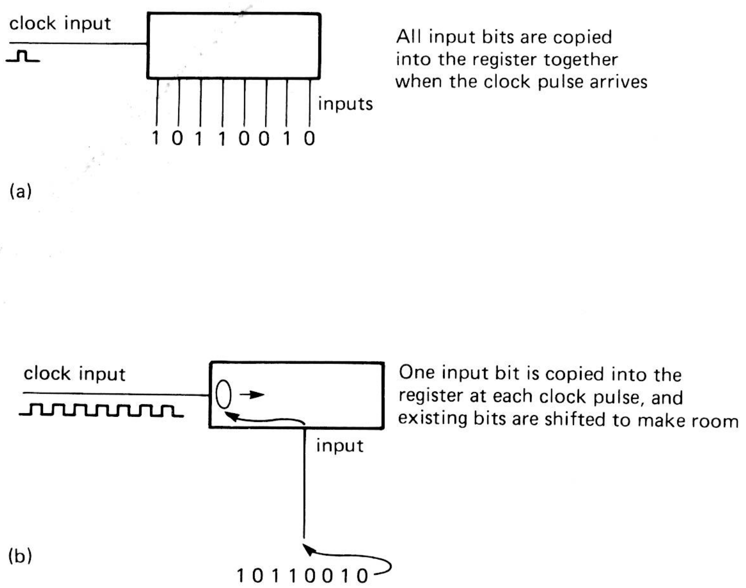

clock input
inputs
1 0 1 1 0 0 1 0
All input bits are copied into the register together when the clock pulse arrives
(a)
clock input
input
1 0 1 1 0 0 1 0
One input bit is copied into the register at each clock pulse, and existing bits are shifted to make room
(b)

Fig. 1.7 Registers, parallel and serial. A parallel register must have as many input/output connections as it stores bits. All of the inputs and outputs will be active at a clock pulse. The serial register deals with 1 bit at each clock pulse, and shifts bits along the register at each clock pulse.

terminal. A register can be loaded with bits, and the loading can be parallel or serial. Taking an 8-bit register as an example, using parallel loading would require eight inputs to the register. When signals (0 or 1) exist on these eight lines, then a pulse to a clock terminal on the register will load in the 8 bits, meaning that the signal levels are stored in the register. If the register is to be serial loaded, then only one input line is used. Each time a bit signal exists on the line, applying a clock pulse will cause the register to load in this bit *and shift all the other bits one place along to make room*. Figure 1.7 illustrates the principles of parallel and of serial loading. Registers can also pass their stored bit signals to other circuits, and once again the methods may be parallel or serial. If parallel output is used, there must be one output for each bit, and the outputs will be connected to the appropriate flip-flops of the registers when the clock pulse is received. If serial output is used, only one output is needed, but only 1 bit will be fed out for each clock pulse.

Registers and gates are the building blocks of both microprocessors and memory chips. A memory chip is a set of registers which is equipped with gates to switch signals in or out as required. The microprocessor is a more complicated device, but in essence it consists also of registers which are connected through gates. This is the important feature that

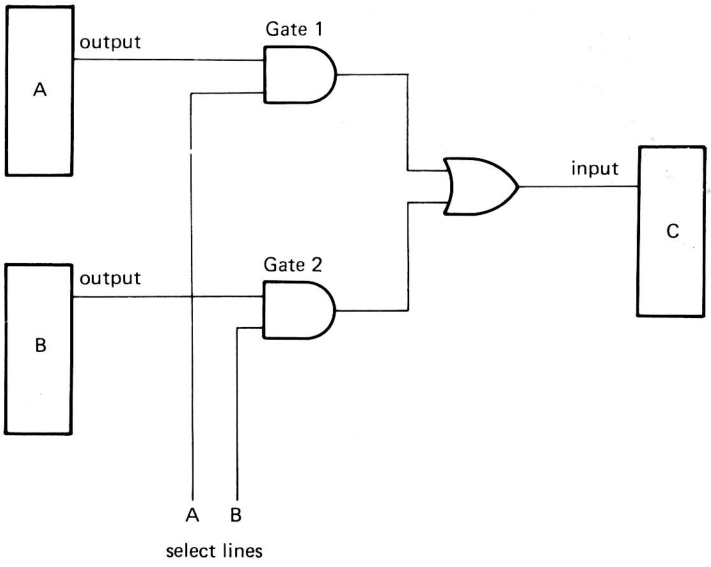


Fig. 1.8 Using gates to change signal paths from one of two registers to a third. The OR gate which is illustrated may not be physically present inside the microprocessor (if, for example, a wired-OR construction is used), but its action will be implemented in the logic.

ultimately makes the microprocessor a programmable device. Imagine, to start with, that we have three 1-bit registers (flip-flops) which are connected by gates as illustrated in Figure 1.8. The outputs of either register A or register B can be connected to register C through the gates. If gate 1 is enabled and gate 2 disabled, then the output of register A is connected to register C, and the bit will be transferred at the next clock pulse. If gate 2 is enabled and gate 1 disabled, then the transfer will be from register B to register C. In this example, the output from the registers has been used to provide the input for another register, but it could equally well have provided the input to another gate. The important point is that the signal path has been controlled by signals to two gates, and if we wanted to control a set of eight signals instead of one, then the same principles apply. The next point is how the gate signals are to be applied. Getting back to the simple single-bit control system, we needed two signals to the gates. For connecting registers A and C, we needed gate 1 enabled, gate 2 disabled. For connecting registers B and C, we needed gate 1 disabled, gate 2 enabled. These two cases require signals of (binary) 10 (A to C) or 01 (B to C) at the gate inputs 1 and 2 respectively. We could disable both gates by using 00, but we'll leave that possibility for the moment.

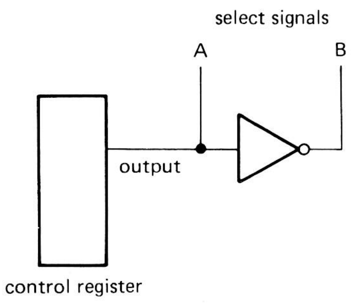


Fig. 1.9 Using a single output of a control register, here illustrated as of 1 bit only, to provide select signals that will control the data path of Figure 1.8.

The next step is to imagine that the gate inputs are provided from a register. The output of the register (Figure 1.9) can be connected to the input of gate 1, and through an inverter to gate 2. In this way, if the output of the register is a 1, then gate 1 is enabled, gate 2 is disabled, and registers A and C are connected. Making the output of this control register equal to zero will reverse the gating, and connect registers B and C. This one simple step has, however, made our simplified circuit into a programmable device! The programming is carried out by storing a bit in the control register, because that bit will then determine the signal paths between the other registers. Programming now consists of placing suitable bits into a control register, so that the gating circuits can then make the correct connections between registers. Though this has been a simplified explanation, all of the principles of operation are identical. For a microprocessor, then, there will be a control register which will be used to contain bits that open or close gates and so make or break connections between other registers.

Before we abandon this simple model of a programmable device, though, we can use it to demonstrate another point about programmable operation. The timing of the operations is very important for any programmable actions. In the simple example, the bit that controls the gates must be in place, in the control register, before the transfer of bits between registers can take place. Because of the type of register construction that is used, this means that the action of transferring a bit from one register to another would require three clock pulses. On the first clock pulse the control register would accept the programming bit. On the second clock pulse the registers A or B would be connected to register C and on the third clock pulse the bit would be transferred. In the simple circuits of Figures 1.8 and 1.9 , of course, there are always register connections between A and C or B and C, whether there is a 1 or a 0 in