Chapter 2
Logic Circuitry

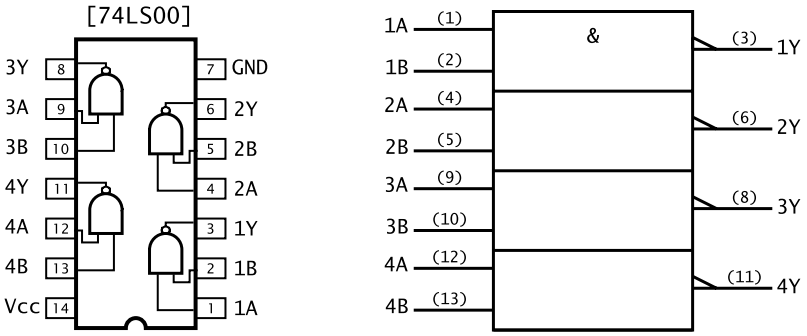
We have noted that digital processing is all about transmission, manipulation and storage of binary word patterns. Here we will extend the concepts introduced in the last chapter as a lead into the architecture of the computer and microcontroller. We will look at some relevant logic functions, their commercial implementations and some practical considerations.

After reading this chapter you will:

- Understand the properties and use of active pull-up, open-collector and 3-state output structures.
- Appreciate the logic structure and function of the natural decoder.
- See how a MSI implementation of an array of XNOR gates can compare two words for equality.
- Understand how a 1-bit adder can be constructed from gates, and can be extended to deal with the addition and subtraction of two n-bit words.
- Appreciate how the function of an ALU is so important to a programmable system.
- Be aware of the structure and utility of a read-only memory (ROM).
- Understand how two cross-coupled gates can implement a R S latch.
- Appreciate the difference between a D latch and a D flip flop.
- Understand how an array of D flip flops or latches can implement a register.
- See how a serial cascade of D flip flops can perform a shifting function.
- Understand how a D flip flop can act as a frequency divide by two, and how a cascade of these can implement a binary count.
- See how an ALU/PIPO register can implement a programmable accumulator processor unit.
- Appreciate the function of a RAM.

The first digital integrated circuits, available at the end of the 1960s, were mainly NAND, NOR and NOT gates. The most popular family of logic functions was the 74 series Transistor Transistor Logic (TTL) introduced by Texas Instruments and soon copied by all the major semiconductor manufacturers. In various forms TTL still represents the de facto standard.

S. Katzen, *The Essential PIC18® Microcontroller,*
Computer Communications and Networks,
DOI 10.1007/978-1-84996-229-2_2, © Springer-Verlag London Limited 2010



(a) DIL package *(b) ANSI/IEC logic symbol*

Fig. 2.1 The 74LS00 quad 2-I/P NAND package

The 74LS00[1] comprises four 2-input NAND gates in a 14-pin package. The integrated circuit (IC) is powered with a 5 ± 0.25 V supply between V_{CC}[2] (usually about 5 V) and GND. The logic outputs are 2.4–5 V for the High state and 0–0.4 V for the Low state. Most IC logic families require a 5 V supply, but 3 V versions are available, and some CMOS implementations can operate with a range of supplies between 3 V and 15 V.

The 74LS00 IC is shown in Fig. 2.1(a) in its dual in-line (DIL) package. Strictly it should be described as a positive-logic quad 2-I/P NAND, as the electrical equivalent for the two logic levels 0 and 1 are Low (L is around ground potential) and High (H is around V_{CC}, usually about 5 V). If the relationship $0 \rightsquigarrow H;\ 1 \rightsquigarrow L$ is used (negative logic) then the 74LS00 is actually a quad 2-I/P NOR gate. The ANSI/IEC[3] logic symbol of Fig. 2.1(b) denotes a Low electrical potential by using the polarity ◺ symbol. The ANSI/IEC NAND symbol shown is thus based on the *real* electrical operation of the circuit. In this case the logic coincides with a positive-logic NAND function. The & operator shown in the top block is assumed applicable to the three lower gates.

The output structure of a 74LS00 NAND gate is **active pull-up**. Here both the High and Low states are generated by connection via a low-resistance switch to V_{CC} or GND, respectively. In Fig. 2.2(a) these switches are shown for simplicity as metallic contacts, but they are of course transistor derived.

[1]The LS stands for "low-power schottky transistor". There are very many other versions, such as ALS (advanced LS), AS (advanced schottky) and HC (high-speed complementary metal-oxide transistor, CMOS). These family variants differ in speed and power consumption, but for a given number designation have the same logic function and pinout.

[2]For historical reasons the positive supply on logic ICs are usually designated as V_{CC}; the C referring to a bipolar's transistor collector supply. Similarly field-effect circuitry sometimes use the designation V_{DD} for drain voltage. The zero reference pin is normally designated as the ground point (GND), but sometimes the V_{EE} (for emitter) or V_{SS} (for source) label is employed.

[3]The American National Standards Institution/International Electrotechnical Commission.

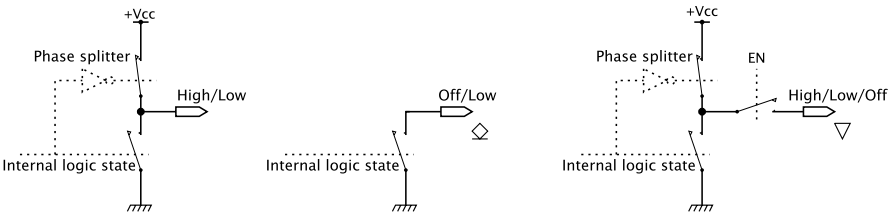


Fig. 2.2 Output structures

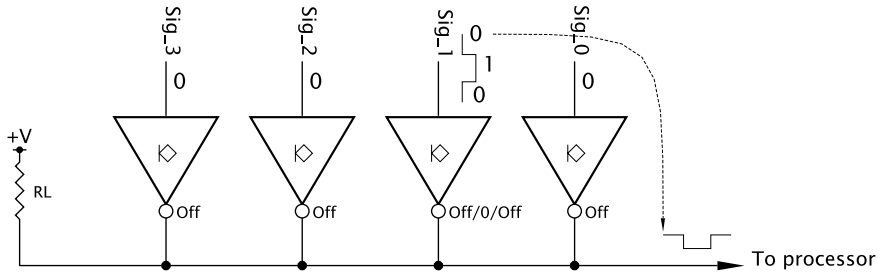


Fig. 2.3 Open-collector buffers driving a party line

Logic circuits, such as the 74LS00, change output state in around 10 nanoseconds.[4] To be able to do this, the capacitance of any interconnecting conductors and other logic circuits' inputs must be rapidly discharged. Mainly for this reason, active pull-up (sometimes called totem-pole) outputs are used by most logic circuits. There are certain circumstances where alternative output structures have some advantages. The **open-collector** (or open-drain) configuration of Fig. 2.2(b) provides a 'hard' Low state, but the High state is in fact an open circuit. The High-state voltage can be generated by connecting an external resistor to either V_{CC} or indeed to a different power rail. Nonorthodox devices, such as relays, lamps or light-emitting diodes, can replace this pull-up resistor. The output transistor is often rated with a higher than usual current and/or voltage rating for such purposes.

The application of most interest to us here is illustrated in Fig. 2.3. Here four open-collector gates share a *single* pull-up resistor. Note the use of the ◇ symbol to denote an open-collector output. Assume that there are four peripheral devices, any of which may wish to attract the attention of the processor, e.g., computer or microcontroller. If this processor has only one Attention pin, then the four Signal lines must be **wire-ORed** together as shown. With all Signal lines inactive (logic 0) the outputs of all buffer NOT gates are off (state H), and the party line is pulled up to +V by RL. If *any* Signal line is activated (logic 1), as in Sig_1, then the output of

[4] A nanosecond is 10^{-9} s, so 100,000,000 transitions each second are possible.

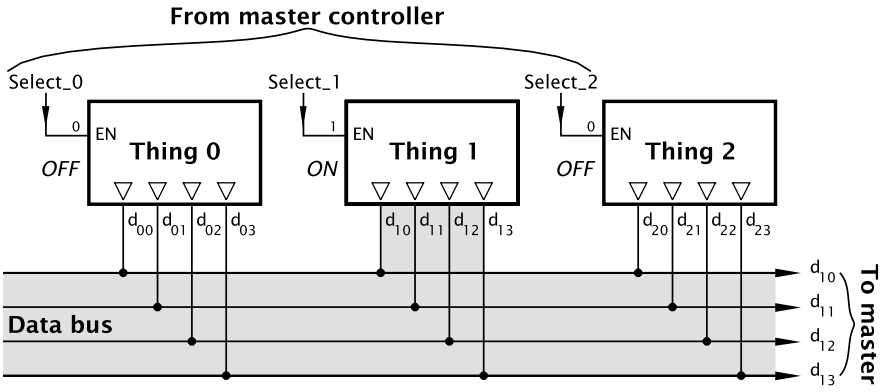


Fig. 2.4 Sharing a bus

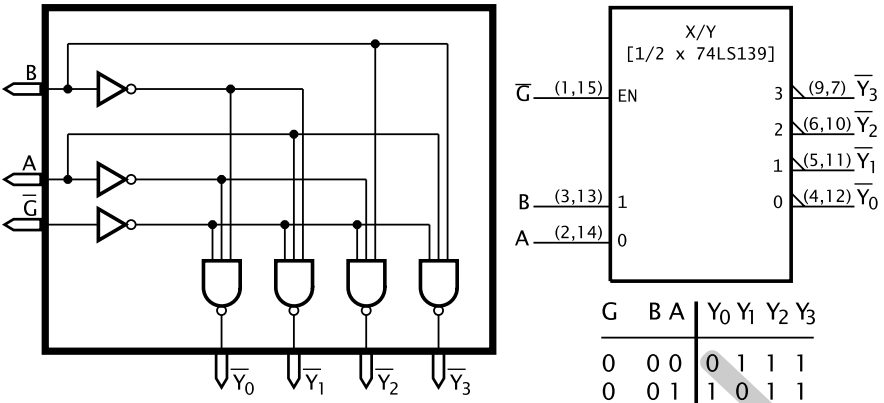
the corresponding buffer gate goes hard Low. This pulls the party line Low and thus interrupts the processor.

The **three-state** structure of Fig. 2.2(c) has the properties of both preceding output structures. When enabled, the two logic states are represented in the usual way by high and low voltages. When disabled, the output is open circuit irrespective of the activities of the internal logic circuitry and any change in input state. A logic output with this three-state is indicated by the $\bigtriangledown$ symbol.

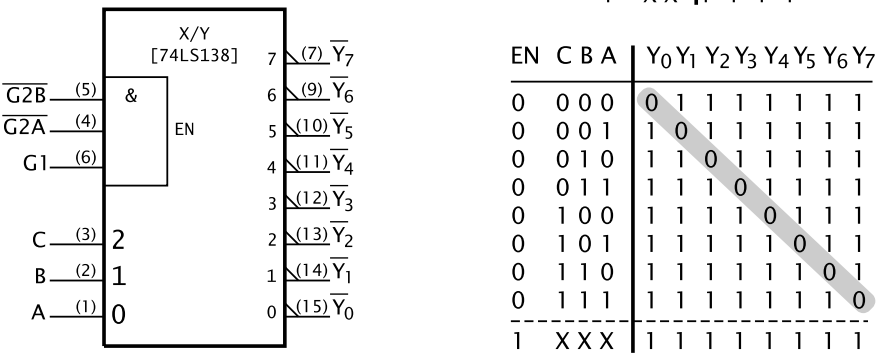
As an example of the use of this structure, consider the situation depicted in Fig. 2.4. Here a master controller wishes to read one of several devices, all connected to this master over a set of party lines. As this data highway or **data bus** is a common resource, only the selected device can be allowed access to the bus at any one time. The access has to be withdrawn immediately after the data has been read, so that another device can use the resource. As shown in the diagram, each 'thing' connected to the bus outputs is designated by the $\bigtriangledown$ symbol. When selected, *only* the active logic levels will drive the bus lines. The 74LS244 octal ($\times 8$) 3-state (sometimes called tristate or TRIS) buffer has high-current outputs (designated by the $\triangleright$ symbol) specifically designed to charge/discharge the capacitance associated with long bus lines.

Integrated circuits with a complexity of up to 12 gates are categorized as small-scale integration (SSI). Gate counts upwards to 100 on a single IC are medium-scale integration (MSI); up to 1000 are known as large-scale integration (LSI) and over this, very large scale integration (VLSI). Memory chips and microcontrollers are examples of this latter category.

The NAND gate networks shown in Fig. 2.5 are typical MSI-complexity ICs. Remembering that the output of a NAND gate is logic 0 only when *all* its inputs are logic 1 (see Fig. 1.2(c) on p. 13) then we see that for any combination of the *select* inputs BA ($2^1\,2^0$) in Fig. 2.5(a) only *one* gate will go to logic 0. Thus output $\overline{Y_2}$ will be activated when BA = 10. The associated truth table shows the circuit *decodes* the binary address BA so that address n selects output $\overline{Y_n}$. The 74LS139 is described as a dual 2- to 4-line **natural decoder**. Dual because there are two



(a) The 74LS139 dual 2- to 4-line decoder



(b) The 74LS138 3- to 8-line decoder

Fig. 2.5 The 74LS138 and 74LS139 MSI natural decoders

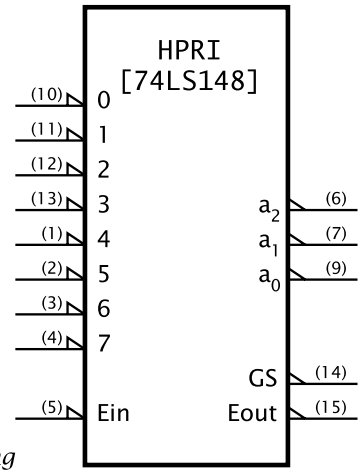
such circuits in the one chip. The symbol X/Y denotes converting code X (natural binary) to code Y (unary—one of n). The enabling input $\overline{\mathsf{G}}$ is connected to all gates in parallel. Thus the decoder function only operates if $\overline{\mathsf{G}}$ is Low (logic 0). If $\overline{\mathsf{G}}$ is High, then irrespective of the state of BA (the X entries in the truth table denote a 'don't care' situation) all outputs remain deselected—logic 1. An example of the use of the 74LS139 is given in Fig. 2.25 on p. 38.

The 74LS138 of Fig. 2.5(b) is similar, but implements a 3- to 8-line decoder function. The state of the three address lines CBA ($2^2\,2^1\,2^0$) n selects only one of the eight outputs $\overline{\mathsf{Y}_n}$. The 74LS138 has three Gate inputs which generate an internal enabling signal $\overline{\mathsf{G2B}} \cdot \overline{\mathsf{G2A}} \cdot \mathsf{G1}$. Only if both $\overline{\mathsf{G2A}}$ and $\overline{\mathsf{G2B}}$ are Low and G1 is High will the device be enabled.

The **priority encoder** illustrated in Fig. 2.6 is a sort of reverse decoder. Bringing one of the eight input lines Low results in the active-Low three-bit binary equivalent appearing at the output. Thus if $\overline{5}$ is Low, then $\overline{\mathsf{a}_2}\,\overline{\mathsf{a}_1}\,\overline{\mathsf{a}_0} = 010$ (active Low 101).

E_{in} 7 6 5 4 3 2 1 0	$a_2a_1a_0$
0 1 1 1 1 1 1 1 0	1 1 1
0 1 1 1 1 1 1 0 X	1 1 0
0 1 1 1 1 1 0 X X	1 0 1
0 1 1 1 1 0 X X X	1 0 0
0 1 1 1 0 X X X X	0 1 1
0 1 1 0 X X X X X	0 1 0
0 1 0 X X X X X X	0 0 1
0 0 X X X X X X X	0 0 0
1 X X X X X X X X	0 0 0

(a) Positive-logic truth table



(b) Logic diagram/pinning

Fig. 2.6 The 74LS148 highest-priority encoder

If more than one input line is active, then the output code reflects the highest. Thus if both $\overline{5}$ and $\overline{3}$ are Low, the output code is still 010. Hence the label `HPRI` for Highest PRIority. The device is enabled when Enable_IN ($\overline{E_{in}}$) is Low. Enable_OUT ($\overline{E_{out}}$) and Group_Strobe ($\overline{GS}$) are used to cascade 74LS148s to expand the number of lines.

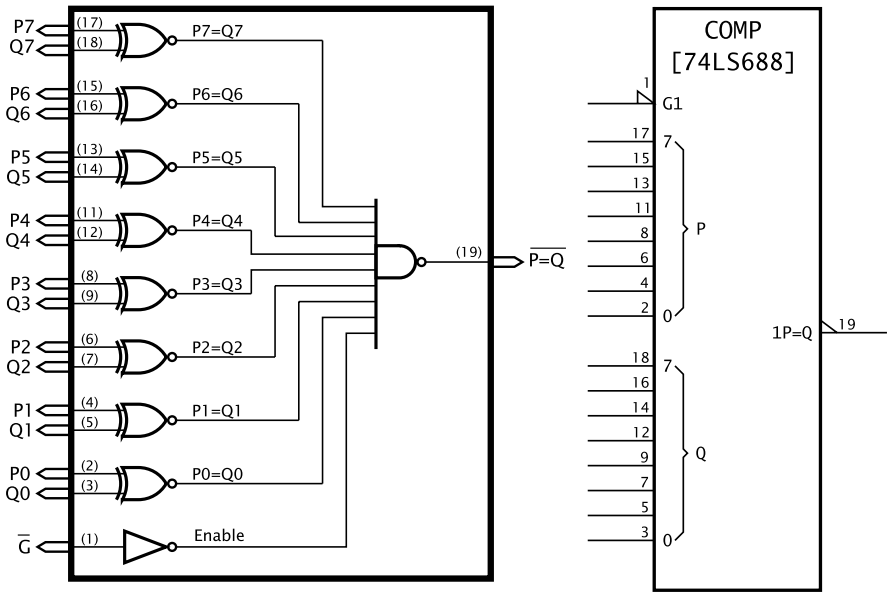
A large class of ICs implement arithmetic operations. The gate array illustrated in Fig. 2.7 detects when the 8-bit byte `P7, ..., P0` is identical to the byte `Q7, ..., Q0`. Eight XNOR gates each give a logic 1 when its two input bits `Pn` & `Qn` are identical, as described on p. 14. Only if *all* 8-bit pairs are the same, will the output NAND gate go Low. The 74LS688 **equality comparator** also has a direct input $\overline{G}$ into this NAND gate, acting as an overall enabling signal.

The ANSI/IEC logic symbol, shown in Fig. 2.7(b), uses the `COMP` label to denote the arithmetic comparator function. The output is prefixed with the numeral `1`, indicating that its operation `P=Q` is dependent on any input qualifying the same numeral; that is `G1`. Thus the active-Low enabling input `G1` gates the active-Low output, `1P=Q`.

One of the first functions beyond simple gates to be integrated into a single IC was that of addition. The truth table of Fig. 2.8(a) shows the sum (S) and carry-out (C_1) resulting from the addition of the two bits A and B and any carry-in (C_0).

For instance, row 6 states that adding two 1s with a carry-in of 0 gives a sum of 0 and a carry-out of 1 ($1 + 1 + 0 = {}^{1}0$). To implement this row we need to detect the pattern 1 1 0; that is, $A \cdot B \cdot \overline{C_0}$; which is gate 6 in the logic diagram. Thus we have by ORing all applicable patterns together for each output:

$$S = (\overline{A} \cdot \overline{B} \cdot C_0) + (\overline{A} \cdot B \cdot \overline{C_0}) + (A \cdot \overline{B} \cdot \overline{C_0}) + (A \cdot B \cdot C_0),$$
$$C_1 = (\overline{A} \cdot B \cdot C_0) + (A \cdot \overline{B} \cdot C_0) + (A \cdot B \cdot \overline{C_0}) + (A \cdot B \cdot C_0).$$



(a) Logic function *(b) ANSI/IEC logic symbol*

Fig. 2.7 The 74LS688 octal equality detector

Using such a circuit for *each* column of a binary addition, with the carry-out from column $k-1$ feeding the carry-in of column k means that the addition of any two n-bit words can be implemented simultaneously.

As shown in Fig. 2.8(b), the 74LS283 adds two 4-bit nybbles in 25 ns. In practice the final carry-out C_4 is generated using additional circuitry to avoid the delays inherent on the carries rippling through each stage from the least to the most significant digit. n 74LS283s can be cascaded to implement addition for words of $4 \times n$ width. Four 74LS283s can perform a 16-bit addition in 45 ns, the extra time being accounted for by the carry propagation between the two units.

Adders can, of course, be coaxed into subtraction by inverting the minuend and adding one, that is 2's complementation—as described on p. 9. An adder/subtractor circuit could be constructed by feeding the minuend word through an array of XOR gates acting as programmable inverters (see p. 14). The mode line $\overline{\text{ADD}}/\text{SUB}$ in Fig. 2.9 that controls these inverters also feeds the Carry-In C_0, effectively adding one when in the Subtract mode.

Extending this line of argument leads to the **arithmetic logic unit** (**ALU**). An ALU is a circuit which can undertake a selection of arithmetic and logic processes on input data as controlled by mode inputs. The 74LS382 in Fig. 2.10 processes two 4-bit operands in eight ways, as controlled by the three Mode Select bits $S_2\,S_1\,S_0$ and tabulated in Fig. 2.10(a). Besides addition and subtraction, the logic operations of AND, OR and XOR are supported. The 74LS382 also generates the 2's complement overflow function—see p. 10.

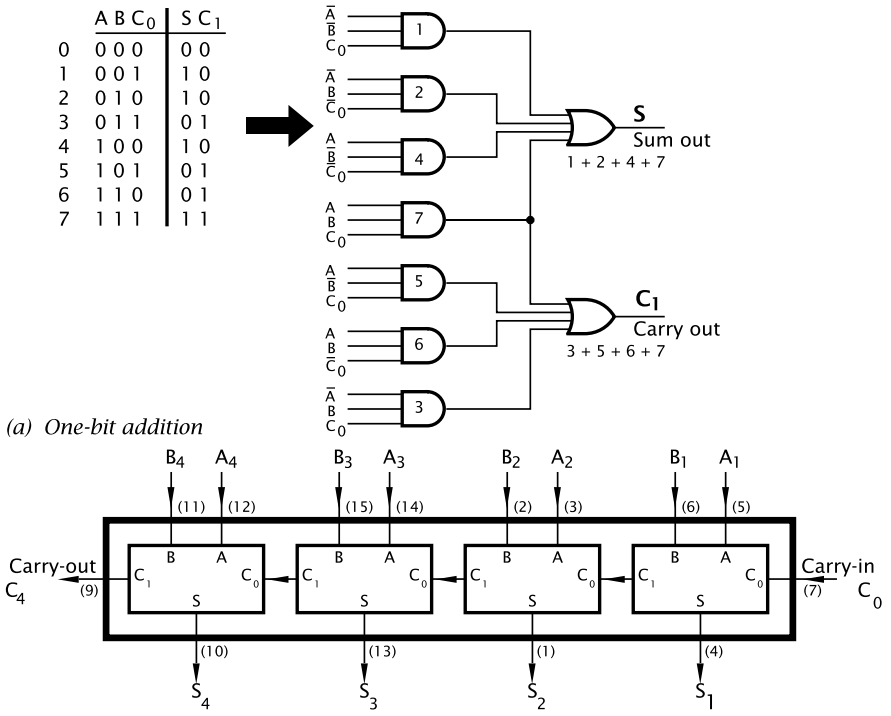


Fig. 2.8 Addition

As we shall see, the ALU is at the heart of the computer and microcontroller architectures. By feeding the Mode Select inputs with a series of binary words, a program of operations can be performed by the ALU. Such **operation codes** are stored in an external memory, and are accessed sequentially by the computer's control circuits.

Sequences of program operation codes are normally stored in some kind of LSI read-only memory. Consider the architecture illustrated in Fig. 2.11. This is essentially a 3- to 8-line decoder driving an 8×2 array of diodes. The 3-bit address selects only row n for each input combination n. If a diode is connected to this row, then it conducts and brings the appropriate column Low. The inverting 3-state output buffer consequently gives a High for each connected diode and Low where the link is broken. The pattern of diode links then defines the output code for each input. For illustrative purposes, the structure has been programmed to implement the 1-bit full adder of Fig. 2.8(a), but *any* two functions of three variables can be generated.

The diode matrix look-up table shown here is known as a **read-only memory** (**ROM**), as its 'memory' is in the diode pattern, which is programmed in when the device is manufactured. Early devices, which were typically decoder/32×8 matrices, usually came in user-programmable versions in which the interconnections were implemented with fusible links. By using a high voltage, a selection of

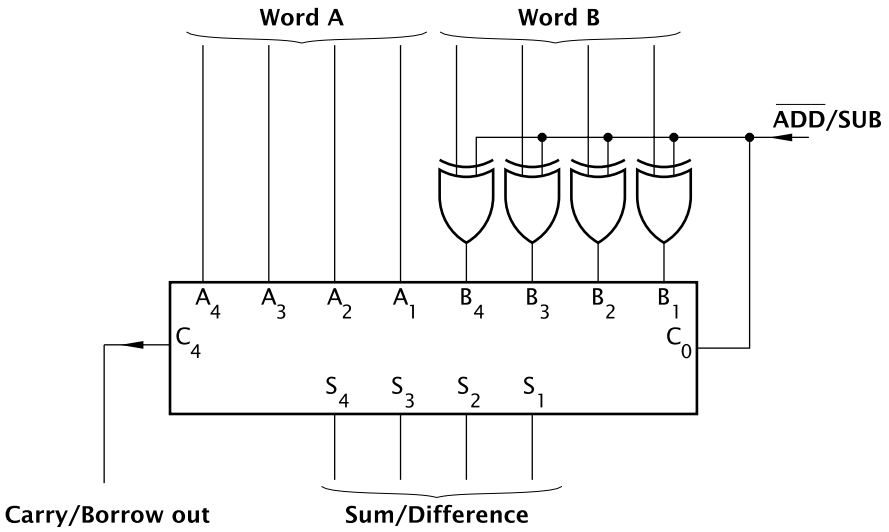
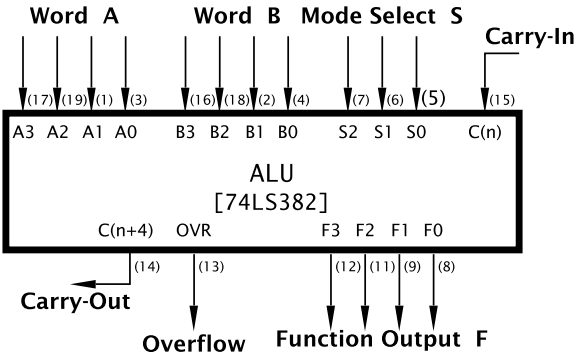


Fig. 2.9 Implementing a programmable adder/subtractor

$S_2S_1S_0$	Operation	
0 0 0	Clear	(F=0000)
0 0 1	Subtract	(B-A)
0 1 0	Subtract	(A-B)
0 1 1	Add	(A+B)
1 0 0	XOR	(A ⊕ B)
1 0 1	OR	(A + B)
1 1 0	AND	(A • B)
1 1 1	Preset	(F=1111)

(a) Function table



(b) Logic diagram/pinning

Fig. 2.10 The 74LS382 ALU

diodes could be taken out of contact. Such devices are called **programmable ROMs** (**PROMs**).

Fuses are messy when implementing the larger sizes of VLSI PROMs necessary to store computer programs. For instance, the small 27C64 PROM shown in Fig. 2.12 has the equivalent of 65,536 fuse/diode pairs, and this is a relatively small device capable of storing 8192 bytes of memory. The 27C64 uses the electrical charge on the floating gate of a metal-oxide field-effect transistor (MOSFET) as the programmable link, with another MOSFET to replace the diode. Charge can be tunneled onto this isolated gate by, again, using a high voltage. Once on the gate, the electric field keeps the link MOSFET conducting. This charge takes many

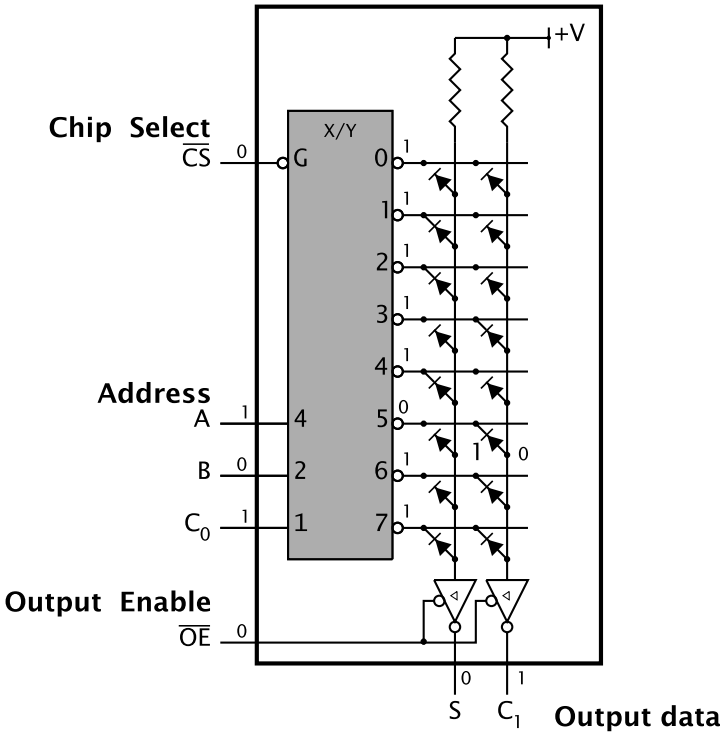
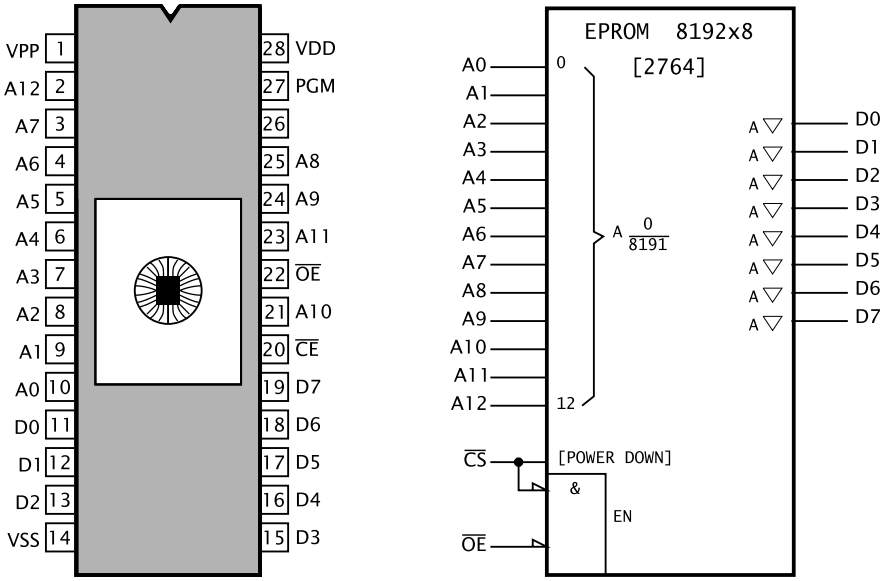


Fig. 2.11 A ROM-implemented 1-bit adder

decades to leak away, but this can be dramatically reduced to about 20 minutes by exposure to intense ultraviolet radiation. For this reason the 27C64 is known as an **erasable PROM** (**EPROM**). When an EPROM is designed for reusability, a quartz window is integrated into the package, as shown in Fig. 2.12 and on p. 2. Programming is normally done externally with special equipment, known as PROM programmers, or colloquially as PROM blasters. Versions without windows are referred to as one-time programmable (OTP) ROMs, as they cannot easily be erased once programmed. They are, however, much cheaper to produce and are thus suitable for small- to medium-scale production runs. However, as a general rule flash EEPROM has a more limited lifetime, as measured as the number of times a cell can be written to.

Figure 2.13 shows a simplified representation of such a floating-gate MOSFET link. The cross-point device is a metal-oxide enhancement *n*-channel field-effect transistor TR1, rather than a diode. This MOSFET has its gate G1 connected to the X line and its source S1 to the Y line. If its drain D1 is connected to the positive supply and the X line is selected (positive), then the Y line too becomes positive (positive-logic 1) as TR1 is conducting (switch is on). However, if TR1 is disconnected from V_{DD} then it does not conduct and the output on the Y line is logic 0. Transistor TR2 is in series with V_{DD} and thus acts as the programmable element.



(a) Dual in-line package *(b) Logic symbol*

Fig. 2.12 The 2764 erasable PROM (EPROM)

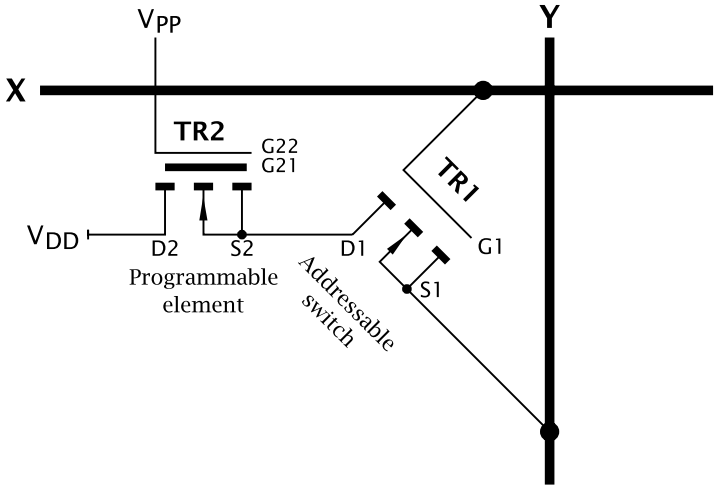


Fig. 2.13 Floating-gate MOSFET link

Transistor TR2 has an extra unconnected gate buried in the silicon dioxide insulation layer. Normally there is no charge on this gate and TR2 is off. If the programming voltage V_{PP} is pulsed high to typically 20–25 V, negative charges tunnel across the

extremely thin insulation surrounding the buried gate. This turns TR2 on permanently and thus connects TR1 to its supply. This shows up as a logic 1 on the Y line when selected by the internal memory decoder.

This charge remains more or less permanently on the buried gate until it is exposed to ultraviolet light. The high-energy light photons knock electrons (negative charges) out of the buried (floating) gate[5] effectively discharging in around 20 minutes and wiping out all stored information.

There are PROM structures which can be erased electrically, often *in situ* in the circuit. These are known variously as electrically-erasable PROMs (EEPROMs) or flash memories. In the former case a large negative pulse at V_{PP} causes the captured electrons on the buried gate to tunnel back out. Generally the negative voltage is generated on the chip, which saves having to provide an additional external supply. The **flash** variant of EEPROM relies on hot electron injection rather than tunneling to charge the floating gate. The geometry of the cell is approximately half the size of a conventional EEPROM cell which increases the memory density. Programming voltages are also somewhat lower. An example of a commercial EEPROM memory is given in Fig. 12.28 on p. 443.

Most modern EPROMs/EEPROMs are fairly fast, taking around 150 ns to access and read. Programming is slow, at perhaps 10 ms per word, but this is an infrequent activity. Flash EEPROMs program around 100 times faster, in around 100 µs per cell. However, as a rule they have a more limited lifetime, as measured by the number of times they can be successfully written to. Typically this may be around 100,000 times[6] as compared to over a million.

All the circuits shown thus far are categorized as **combinational logic**. They have no memory in the sense that the output depends only on the present input, and not the sequence of events leading up to that input. Logic circuits, such as latches, counters, registers and read/write memories are described as **sequential logic**. Their output not only depends on the current input, but the sequence of prior inputs.

Consider a typical doorbell pushswitch. When you press such a switch the bell rings, and it stops as soon as you release it. This switch has no memory.

Compare this with a standard light switch. Set the switch and the light comes on. Moreover, it remains on when you remove the stimulus (usually your finger!). To turn the light off you must reset the switch. Again it remains off when the input is taken away. This type of switch is known as a **bistable**, as it has two stable states. Effectively it is a 1-bit memory cell, that can store either an on or off state indefinitely.

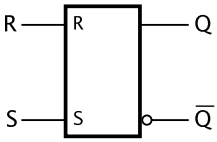
A read/write memory, such as the 6264 device of Fig. 2.26, implements each bistable cell using two cross-coupled transistors. Here we are not concerned with

[5] This is called the Einstein effect. Einstein was awarded his Nobel prize for this discovery and not for his theories of relativity, as these were considered too revolutionary!

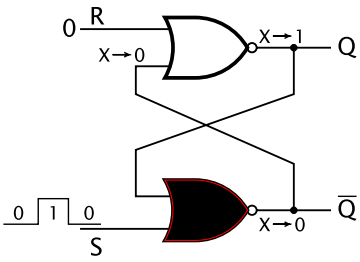
[6] There are around 600,000 seconds in a week and so if a cell is written into once every six seconds the entire lifetime could be used up in a week!

R S	Q
0 0	Q (no change)
0 1	1 (set)
1 0	0 (reset)

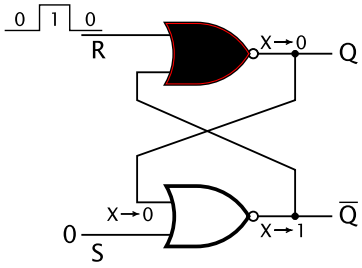
(a) Defining RS latch truth table



(b) Logic symbol with true/complement outputs



(c) Setting the latch



(d) Resetting the latch

Fig. 2.14 The R S latch

this microscopic view. Instead, consider the two cross-coupled NOR gates of Fig. 2.14. Remembering from Fig. 1.3(c) on p. 13 that any logic 1 into a NOR gate will always give a logic 0 output irrespective of the state of the other inputs, allows us to analyse the circuit:

- If the S input goes to 1, then output $\overline{\mathsf{Q}}$ goes to 0. Both inputs to the top gate are now 0 and thus output Q goes to 1. If the S input now goes back to 0, then the lower gate remains 0 (as the Q feedback is 1) and the top gate output also remains unaltered. Thus the latch is *set* by pulsing the S input.
- If the R input goes to 1, then output Q goes to 0. Both inputs to the bottom gate are now 0 and thus output $\overline{\mathsf{Q}}$ goes to 1. If the R input now goes back to 0, then the upper gate remains 0 (as the $\overline{\mathsf{Q}}$ feedback is 1) and the bottom gate output also remains unaltered. Thus the latch is *reset* by pulsing the R input.

In the normal course of events—that is assuming that the R and S inputs are not both active at the same time[7]—then the two outputs are always complements of each other, as indicated by the logic symbol of Fig. 2.14(b).

There are many bistable implementations. For example, replacing the NOR gates by NAND gives a $\overline{\mathsf{R}}\,\overline{\mathsf{S}}$ latch, where the inputs are active on a logic 0. The circuit illustrated in Fig. 2.15 shows such a latch used to debounce a mechanical switch. Manual

[7] If they were, then both Q and $\overline{\mathsf{Q}}$ would go to 0. On relaxing the inputs, the latch would end up in one of its stable states, depending on the relaxation sequence. The response of a latch to a simultaneous Set and Reset input signal is not part of the latch definition, shown in Fig. 2.14(a), but depends on its implementation. For instance, trying to turn a light switch on and off together could end in splitting it in two!

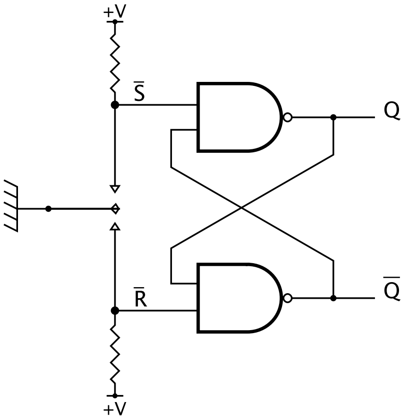


Fig. 2.15 Using a $\overline{R}\,\overline{S}$ latch to debounce a switch

switches are frequently used as inputs to logic circuits. However, most metallic contacts will bounce off the destination contact many times over a period of several tens of milliseconds before settling. For instance, using a mechanical switch to interrupt a computer/microcontroller will give entirely unpredictable results.

In Fig. 2.15, when the switch is moved up and hits the contact the latch move into its Set state. When the contact is broken, the latch remains unchanged, provided that the switch does not bounce all the way back to the lower contact. The state will remain Set no matter how many bounces occur. By symmetry, the latch will reset when the switch is moved to the bottom contact, and remain in this Reset state on subsequent bounces.

The **D latch** is an extension to the R S latch, where the output follows the D (Data) input when the C (Control) input is active (logic 1 in our example) and freezes when C is inactive. The D latch can be considered to be a 1-bit memory cell where the datum is retained at its value at the end of the sample pulse.

In Fig. 2.16(b) the dependency of the Data input with its Control signal is shown by the symbols `C1` and `1D`. The `1` prefix to `D` shows that it depends on any signal with a `1` suffix, in this case the `C` input. That is, `C1`, clocks in the `1D` data.

A flip flop is also a 1-bit memory cell, but the datum is only sampled on an *edge* of the control (known here as the **Clock**) input.

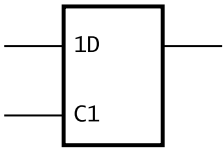
The **D flip flop** described in Fig. 2.16(c) is triggered on a _/‾ (as illustrated in the truth table as ↑), but ‾_ clocked flip flops are common. The edge-triggered activity is denoted as > on a logic diagram, as shown in Fig. 2.16(d).

The 74LS74 shown in Fig. 2.17 has two D flip flops in the one SSI circuit. Each flip flop has an overriding Reset ($\overline{R}$) and Set ($\overline{S}$) input, which are asynchronous, that is, not controlled by the Clock input. MSI functions include arrays of four, six and eight flip flops all sampling simultaneously with a common Clock input.

The 74LS377 shown in Fig. 2.18 consists of eight D flip flops all clocked by the same single Clock input `C`, which is gated by input $\overline{\texttt{G}}$. Thus the 8-bit data `8D`, ..., `1D` is clocked in on the _/‾ of `C` if $\overline{\texttt{G}}$ is Low. In the ANSI/ISO logic diagram shown

C	D	Q	
1	0	0	(transparent)
1	1	1	
0	X	Q	(freeze)

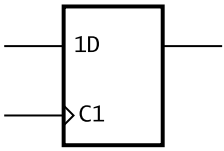
(a) D latch truth table



(b) D latch logic diagram

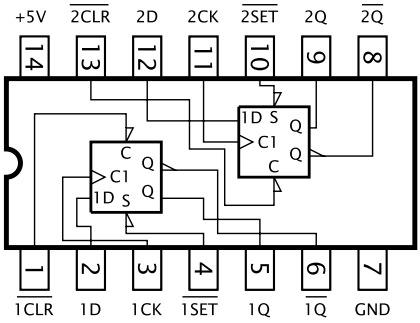
C	D	Q	
↑	0	0	(sample)
↑	1	1	
0	X	Q	
1	X	Q	(hold)
↓	X	Q	

(c) D flip flop truth table

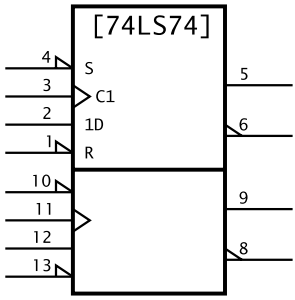


(d) D flip flop logic diagram

Fig. 2.16 The D latch and flip flop



(a) Logic function

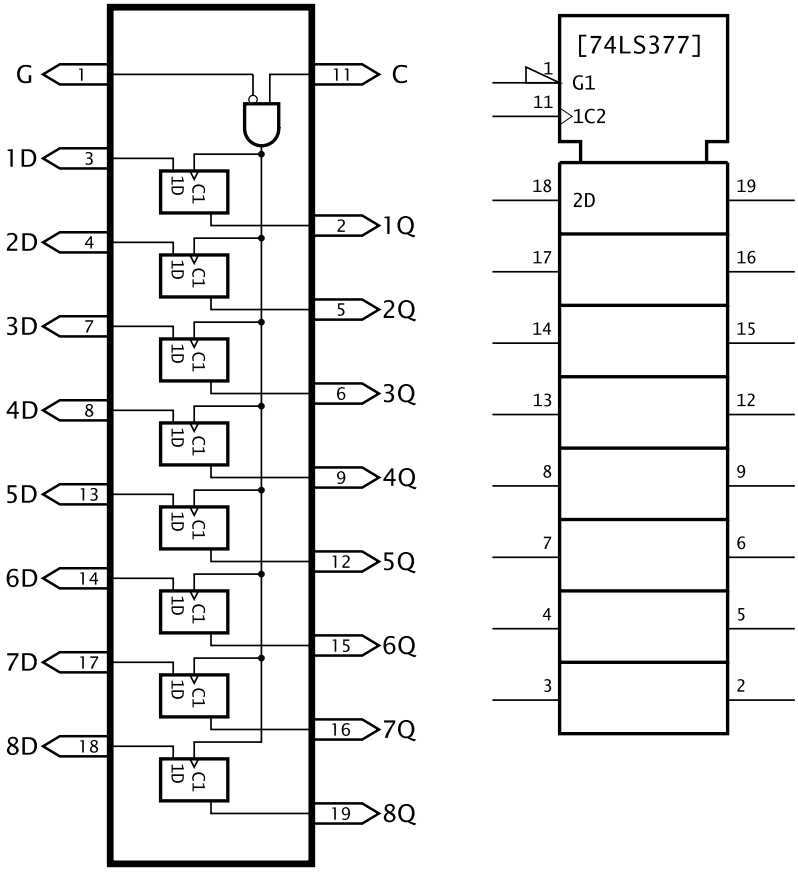


(b) ANSI/IEC logic symbol

Fig. 2.17 The 74LS74 dual D flip flop

in Fig. 2.18(b), this dependency is indicated as G1→1C2→2D, which states that $\overline{\text{G}}$ enables the Clock input, which in turn acts on the Data inputs.

Arrays of D flip flops are known as **registers**, that is, read/write memories that hold a single word. The 74LS377 is technically known as a parallel-in parallel-out (PIPO) register, as data is entered in parallel (that is, all in one go) and is available to read at one go. D latch arrays are also available, such as the 74LS373 octal PIPO register shown in Fig. 2.19, in which the eight D flip flops are replaced by D latches. In addition, the latch outputs have a 3-state capability. This is useful if data is to be captured and later put onto a common data bus to be read subsequently as desired by a computer.

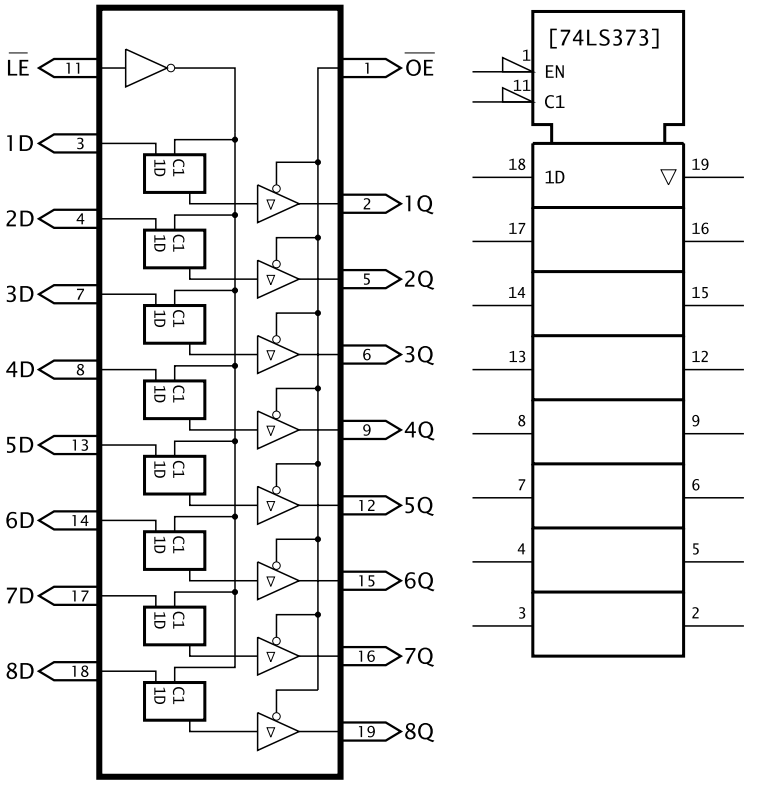


(a) Logic function *(b) ANSI/IEC logic symbol*

Fig. 2.18 The 74LS377 octal D flip flop array

A pertinent example of the use of a PIPO register is shown in Fig. 2.20. Here an 8-bit ALU is coupled with an 8-bit PIPO register, accepting as its input the ALU output, and in turn feeding one input word back to the ALU. This register accumulates the outcome of a series of operations, and is sometimes called an **Accumulator** or **Working register**. To describe the operation of this circuit, consider the problem of adding two words A and B. The sequence of operations, assuming the ALU is implemented by cascading two 74LS382s, might be:

1. Program step.
   - Mode = 000 (Clear).
   - Pulsing Execute loads the ALU output (0000 0000) into the register.
   - Data out is zero (0000 0000).



(a) Logic function *(b) ANSI/IEC logic symbol*

Fig. 2.19 The 74LS373 octal D latch array

2. Program step.

   - Fetch word A down to the ALU input.
   - Mode = 011 (Add).
   - Pulse ⎍ Execute to load the ALU output (word A + zero) into the register.
   - Data out is word A.

3. Program step.

   - Fetch word B down to the ALU input.
   - Mode = 011 (Add).
   - Pulse ⎍ Execute to load the ALU output (word B + word A) into the register.
   - Data out is word B plus word A.

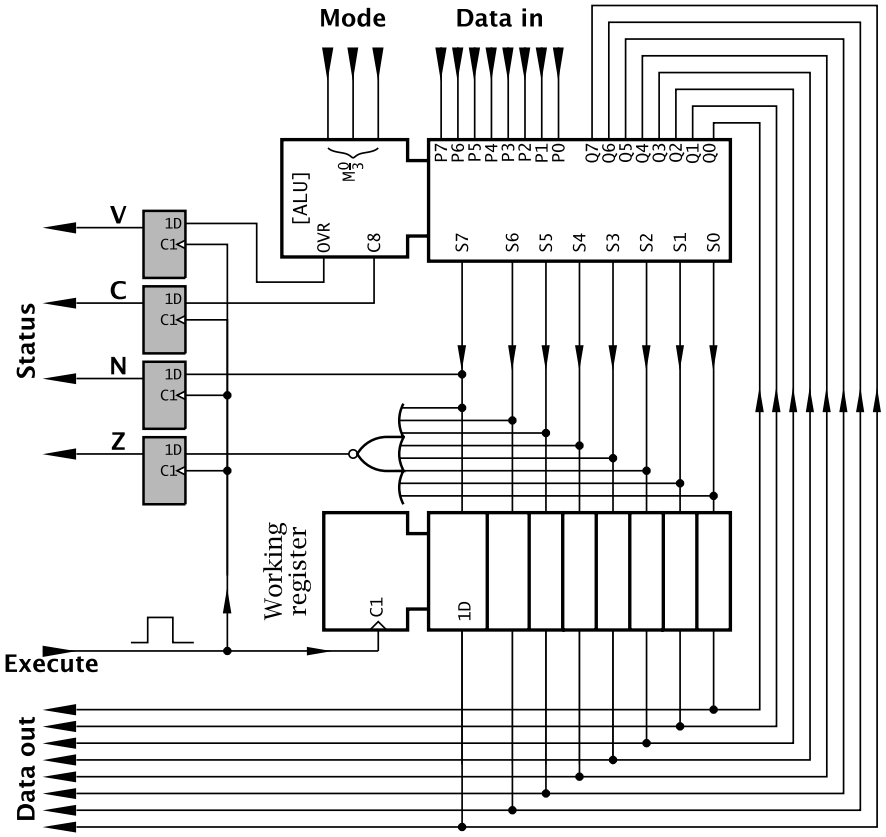


Fig. 2.20 An 8-bit ALU-accumulator processor

The sequence of operation codes, that is 000–100–100 constitutes the program. In practice each instruction would also contain the address (where relevant) in memory of the data to be processed; in this case the locations of word A and word B.

Each outcome of a process will have associated properties. For instance, it may be zero, be negative (most-significant bit is 1), have a carry-out or 2's complement overflow.

Such properties may be significant in the future progress of the program. In the diagram, four D flip flops, clocked by Execute, are used to grab this status information. In this situation the flip flops are usually known as **flags** (or sometimes semaphores). Thus we have **C**, **N**, **Z** and **V** flags, which form a Status register.

As we will see in the next chapter, the ALU/Working register processor is the heart of digital computing engines. In complex systems, such as a computer or microcontroller, the detail of a diagram like Fig. 2.20 is not necessary and will hide the underlying system process from the observer. Figure 2.21 shows the same process at a higher level of abstraction. For instance, the various multiple wire data connections or **buses** are shown as a single thick path; the actual details are unimportant. The

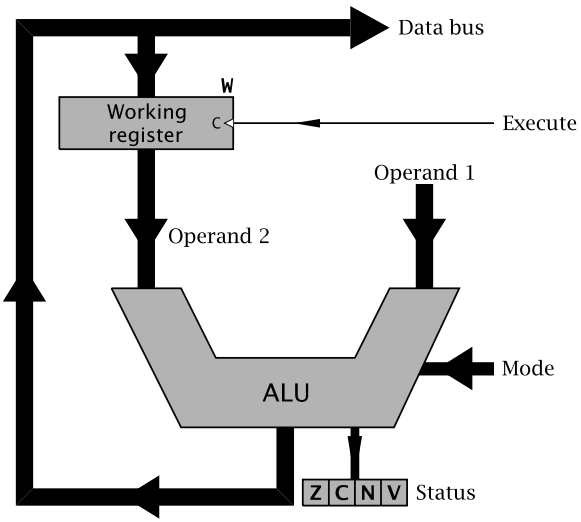


Fig. 2.21 A system look at our ALU-accumulator processor

number of connections in a path is not shown, but if important, is usually indicated by a diagonal tick, thus $\overset{8}{\diagup}$.

The ALU, with its distinctive shape, is at the center of our system. Its two data inputs, or operands, are processed according to the Mode input. Operand 1 comes from outside our system, whilst Operand 2 is connected from the Working register. In a computer, the Mode input codes normally come from the program memory, whilst Operand 1 is obtained from the data memory.

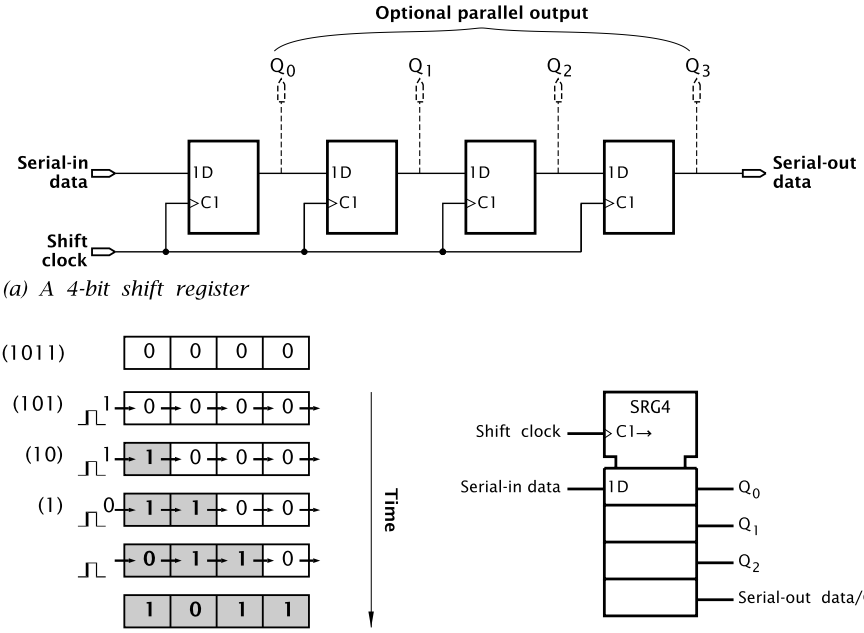
The ALU output can be either latched back into the Working register when sampled by the Execute signal, or it can be fed outside into a data memory via the bus. This enhancement is shown in Fig. 3.2 on p. 43.

There are various other forms of register. The 4-bit **shift register** of Fig. 2.22(a) is an example of a serial-in serial-out (SISO) structure. In this instance the data held in the nth D flip flop is presented to the input of the $(n+1)$th stage. On receipt of a clock pulse (or shift pulse in this context), this data moves into this $(n+1)$th flip flop, i.e., effectively moving from stage n to stage $n+1$. As all flip flops are clocked simultaneously, the entire word moves once to the right on each shift pulse.

In the example of Fig. 2.22 a 4-bit external data nybble is fed into the leftmost stage bit-by-bit as synchronized by the clock. After four shift pulses the serial 4-bit word is held in the register. To get it out again, four further shifts move the word bit-by-bit out of the shift register; this is SISO. If the individual flip flops are accessible then the data can be accessed at one go, that is, serial-in parallel-out.

The logic diagram of Fig. 2.22(b) uses the $\rightarrow$ symbol prefixed by the clock input to indicate the shift action; i.e., `C1` $\rightarrow$. `SRG4` indicates a Shift ReGister 4-stage architecture. An example of an 8-stage shift register is given in Fig. 12.2 on p. 381.

Other architectures include parallel-in serial-out, which is useful for parallel to serial conversion. Counting registers (counters) increment or decrement on each



(a) A 4-bit shift register

(b) Shifting 1011 into the register

(c) The ANSII/IEC symbol for a SIPO register

Fig. 2.22 The SISO shift register

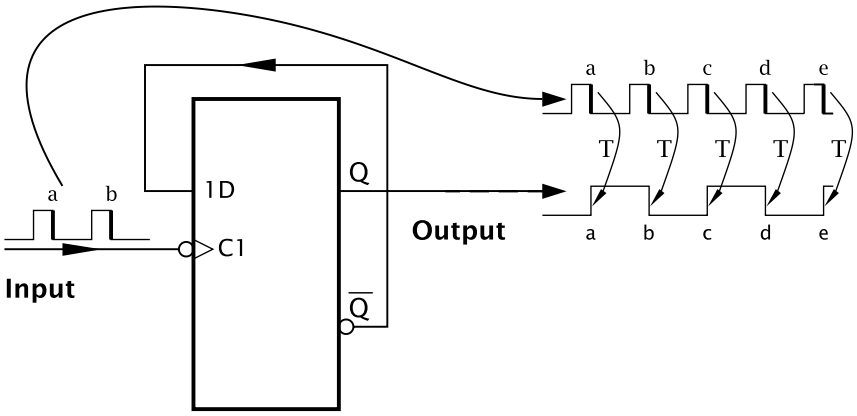
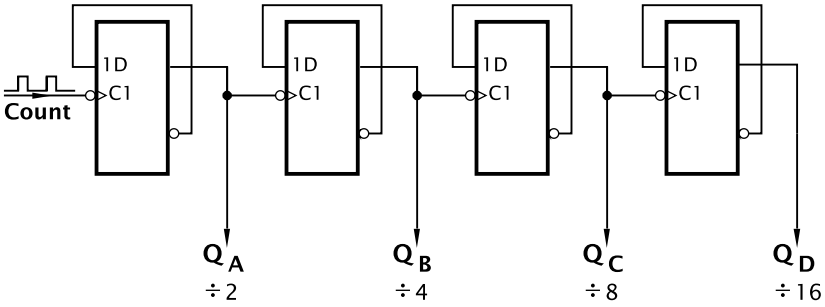


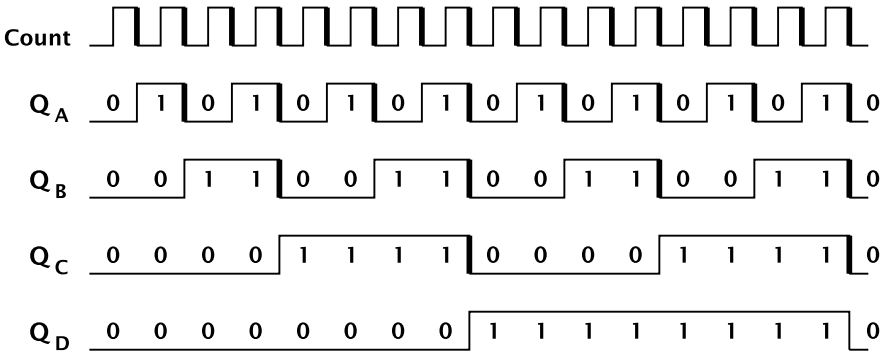
Fig. 2.23 The T flip flop

clock pulse, according to a binary sequence. Typically an n-bit counter can perform a count of 2^n states. Some can also be loaded in parallel and thus act as a store.

Consider the negative-edge triggered D flip flop shown in Fig. 2.23 where its $\overline{\mathtt{Q}}$ output is connected back to the 1D input. On each ⎺⎺_ at the Clock input C1,



(a) Cascading toggle flip flops



(b) Resulting waveforms

Fig. 2.24 A modulo-16 ripple counter

the data at the `1D` input will be latched in to appear at the `Q` output. As it is the complement of this output that is fed back to the input, then the next time the flip flop is clocked the *opposite* logic state will be latched in. This constant alternation is called *toggling* and is depicted on the diagram by `T`. The output waveform resulting from a constant-frequency input pulse train is half this frequency. This waveform is a precision squarewave, provided that the input frequency remains constant. This **T flip flop** is sometimes known as a binary or a divide-by-2.

T flip flops can be cascaded, as shown in Fig. 2.24(a). Here four ⎺_ triggered flip flops are chained, with the output of binary n clocking binary $n+1$. Thus if the input Count frequency was 8 kHz, then Q_A would be a 4 kHz square waveform and similarly Q_B would measure in at 2 kHz, Q_C at 1 kHz, Q_D at 500 Hz.

The waveform Q_A of Fig. 2.24(b) was derived in the same manner as in Fig. 2.23. Q_B is toggled on each ⎺_ of Q_A and likewise for the subsequent outputs. Marking a High as logic 1 and a Low as logic 0 gives the 2^4 (16) positive-logic binary patterns as time advances, with the count rolling over back to state 0 on a continual basis. Each pattern remains in the register until the next event clocks the chain; an event being defined in our example as a ⎺_ at Count. Examining the sequence shows

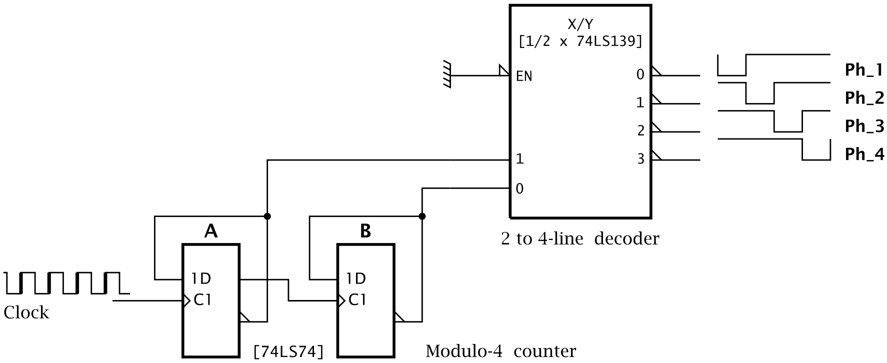


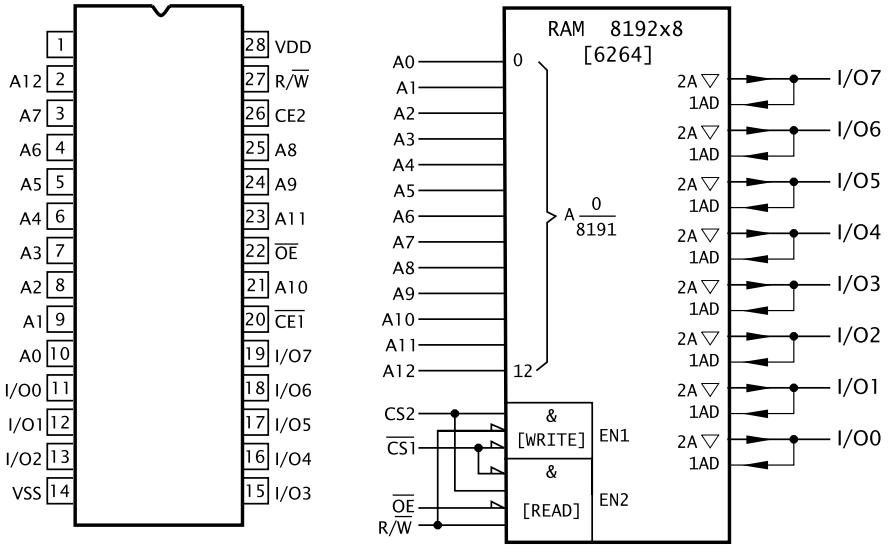
Fig. 2.25 Generating timing waveforms

it to be a natural 8-4-2-1 binary up count, incrementing from b'0000' to b'1111'. In fact, the circuit is a modulo-16 **binary counter** or **timer**. A modulo-*n* count is the sequence taking only the first *n* numbers into account.[8]

In theory there is no limit to the number of stages that can be cascaded. Thus using eight T flip flops would give a modulo-256 (2^8) counter. In practice there is a small propagation delay through each stage and this limits the ultimate frequency. For instance, the 74LS74 dual D flip flop of Fig. 2.17 has a maximum propagation from an event at its Clock input to Q output of 25 ns (the maximum toggling frequency for a single stage, such as in Fig. 2.23, is quoted as 25 MHz). An 8-stage counter thus has a maximum ripple-through time of 200 ns (8×25). If such a **ripple counter** were clocked at the resulting 5 MHz ($\frac{1}{200\ \text{ns}}$) then no sooner would one particular code pattern stabilize then the next one would begin to appear. This is only really a problem if the various states of the counter are to be decoded and used to control other logic. The decoding logic, such as shown in Fig. 2.25, may inadvertently respond to these short transient states and cause havoc. In such cases more sophisticated synchronous counter configurations are more applicable where the flip flops are clocked simultaneously and steered by the appropriate logic configuration to count in the desired sequence.

The circuit illustrated here implements an up count. If the complement $\overline{\text{Q}}$ lines are used as the outputs, but with the clocking arrangements remaining the same, then the count sequence will decrement, that is a down count. Likewise, if ⁄ triggered flip flops, such as the 74LS74 dual flip flop (see Fig. 2.25), are used as the storage element, then the count will be down. It is easily possible to use some simple logic to combine the two functions to produce a programmable up/down counter. It is also feasible to provide logic to load the flip flop array in parallel with any number

[8]Mathematically any number can be converted to its modulo-*n* equivalent by dividing by *n*. The remainder, or modulus, will be a number from 0 to $n - 1$.



(a) Dual in-line package *(b) Logic symbol*

Fig. 2.26 The 6264 8196 × 8 RAM

and then count up or down from that point. Such an arrangement can be thought of as a parallel-in counting register.

In addition to the more obvious uses of a counter register to add up the number of events, such as cans of peas coming along a conveyor belt, there are other uses. One of these is to time a sequence of operations. In Fig. 2.25 a modulo-4 counter is used to address one section of a 74LS139 2- to 4-line decoder; see Fig. 2.5(a). This detects each of the four states of the counter, and the outcome is four time-separated outputs that can be used to sequence, say, the operation of a computer's control section logic—such as that in Fig. 4.5 on p. 76. As a practical point, the complement $\overline{Q}$ flip flop outputs have been used to address the decoder to compensate for the $\_/\overline{\ }$ triggered action that would normally give a down count. Larger counters with the appropriate decoding circuitry can be used to generate fairly sophisticated sequences of control operations.

The term register is commonly applied to a read/write memory that can store a single binary word, typically 4–64 bits. Larger memories can be constructed by grouping n such registers and selecting one of n. Such a structure is sometimes known as a register file. For example, the 74LS670 is a 4 × 4 register file with a separate 4-bit data input and data output and separate 2-bit address. This means that any register can be read at any time, independently of any concurrent writing process.

Larger read/write memories are customarily known as **read/write random-access memories**, or **RAMs** for short. The term random-access indicates that any memory word may be selected with the same access time, irrespective of its position

in the memory matrix.[9] This contrasts with a magnetic tape memory, where the reel must be wound to the sector in question—and if this is at the end of the tape ....

For our example, Fig. 2.26 shows the 6264 RAM. This has a matrix of 65,536 (2^{16}) bistables organized as an array of 8192 (2^{13}) words of 8 bits. Word n is accessed by placing the binary pattern of n on the 13-bit Address pins A12, ..., A0.

When in the Read mode (Read/$\overline{\text{Write}} = 1$), word n will appear at the eight data outputs (I/O7, ..., I/O0) as determined by the state n of the address bits. The A symbol at the input/outputs (as was the case in Fig. 2.12) indicates this addressability. In order to enable the 3-state output buffers, the $\overline{\text{Output Enable}}$ input must be Low.

The addressed word is written into if R/$\overline{\text{W}}$ is Low. The data to be written into word n is applied by the outside controller to the eight I/O pins. This bidirectional traffic is a feature of computer buses.

In both cases, the RAM chip as a whole is enabled when $\overline{\text{CS1}}$ is Low and CS2 is High. Depending on the version of the 6264, this access from enabling takes around 100–150 ns. There is no upper limit to how long the data can be held, provided power is maintained. For this reason, the 6264 is described as static (SRAM). Rather than using a transistor pair bistable to implement each bit of storage, data can be stored as charge on the gate-source capacitance of a single field-effect transistor. Such charge leaks away in a few milliseconds, so needs to be refreshed on a regular basis. Dynamic RAMs (DRAMs) are cheaper to fabricate than SRAM equivalents and obtainable in larger capacities. They are usually found where very large memories are to be implemented, such as found in a personal computer. In such situations, the expense of refresh circuitry is more than amortized by the reduction in cost of the memory devices.

Both types of read/write memories are volatile, that is, they do not retain their contents if power is removed. Some SRAMs can support existing data at a very low holding current and lower than normal power supply voltage. Thus a backup battery can be used in such circumstances to keep the contents intact for many months. The advantage of this strategy over EEPROM technology is the unlimited number of writes to memory—see Footnote 2.

[9] Strictly speaking, ROMs should also be described as random access, but custom and practice has reserved the term for read/write memories.

http://www.springer.com/978-1-84996-228-5

The Essential PIC18® Microcontroller
Katzen, S.
2010, XII, 612 p., Hardcover
ISBN: 978-1-84996-228-5