Chapter 11
Animation and Rotation of Complex Figures

Figure animation and rotation belong to a class of special effects that is seldom critical to the operation of a video game system. These effects can, however, lend a special touch of interest that heightens the players' sense of reality.

The circuits suggested in this chapter can actually be applied without reference to any particular game. Most experimenters will agree that building the complex figures in Chapter 4 offered a unique opportunity to create some fascinating images on the screen. Now there is the chance to add the dimension of motion, animation and rotation, to them.

Figure rotation is a special class of motion calling for rotating the figure on the screen about an imaginary axis extending at right angles to the plane of the screen—in and out of the screen. One of the more common figure rotation effects can be found in connection with a popular commercial combat game where two tanks chase each other around the screen. The tank figure moves only in a forward direction; the player has the ability to rotate the figure, making it possible to move it in virtually any direction.

This figure-rotation feature is actually a form of animation, but as demonstrated later in this chapter, it is a form of figure animation and motion that calls for an unusually high degree of system planning.

An experimenter cannot hope to master the fundamentals of figure animation and rotation without first getting a complete understanding of how complex figures are built in a matrix format. Readers

who feel they are not prepared to design animated figures would do well to study the principles outlined in Chapter 4 once again.

FIGURE ANIMATION

The basic idea behind figure animation (and rotation as well) is to present a series of figures on the screen in a fashion that creates the illusion of motion. The scheme might consist of four different complex figures, each differing from the others in some peculiar way. And when they are flashed onto the screen in a relatively rapid sequence, the observer gets the impression of motion or animation.

The idea is identical to filmed animation. In this video situation, however, the length and complexity of the animated sequence is necessarily limited by the cost and complexity of the circuitry involved.

So the following discussion presents the basic formula for generating a simple animated sequence. While the examples are kept on a rather simple level, the basic scheme can be extended indefinitely, or at least as far as time, money, and patience can carry it.

One of the principle IC devices required for generating an animated sequence is a dual 1-line-to-4-line decoder, 74155. This particular device has been specified in earlier game systems, but it is not important to understand its modes of operation in greater detail.

Figure 11-1 shows the 74155 device, first in a functional block diagram form (Fig. 11-1a), then the pinout (Fig. 11-2b), and finally in a truth-table form (Fig. 11-1c).

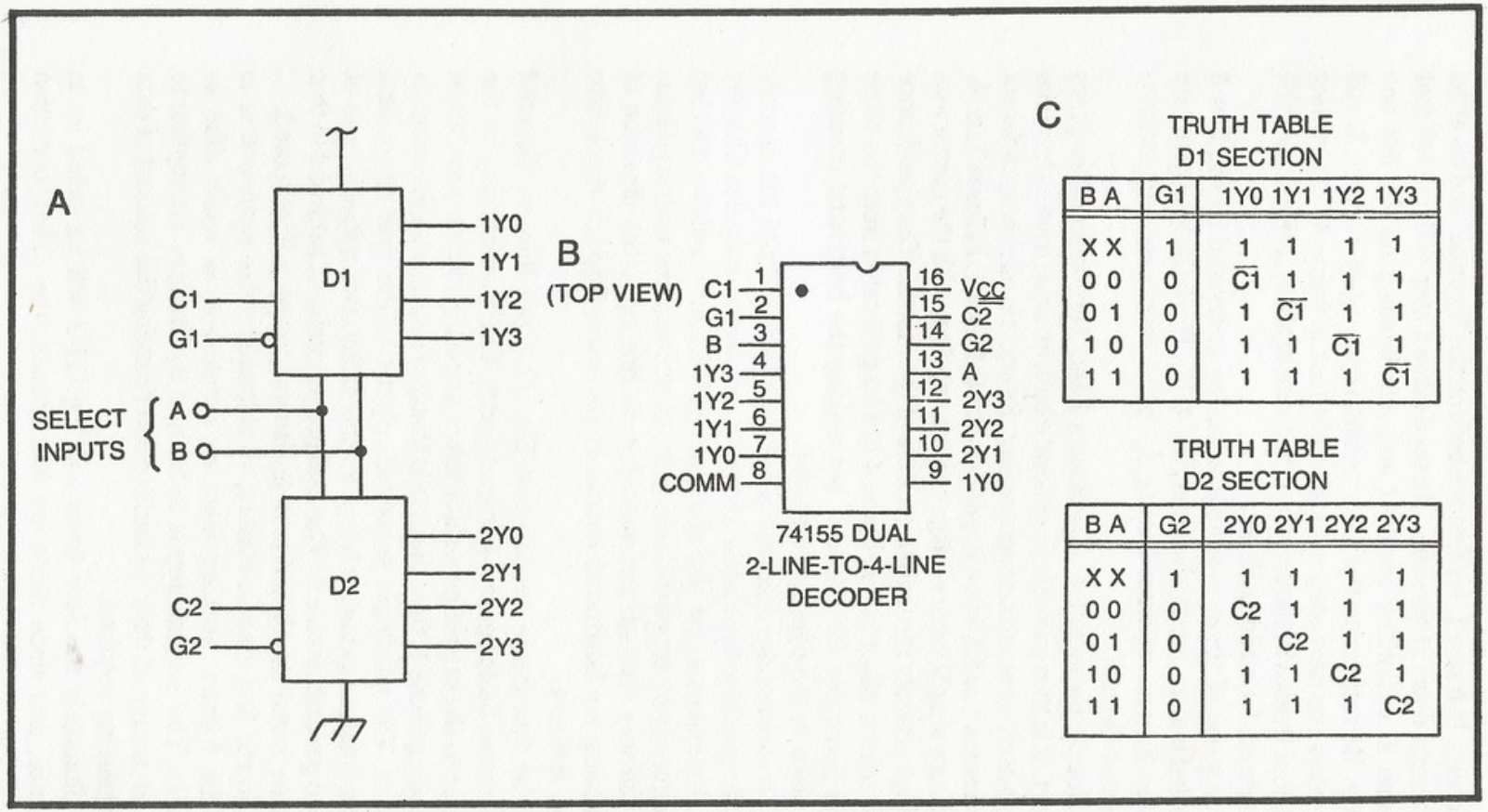
The device is divided into two separate sections as shown in Fig. 11-1a. As illustrated in the truth tables, any logic level present at a C input is delivered to one of the four outputs of each section, depending on the status of select inputs A and B.

Suppose, for example, A=0 and B=1. This select status selects output Y2 for both sections, delivering an inverted version of C1 to output 1Y2 and a noninverted version to output 2Y2. A given C input can appear at one, and only one, of its respective Y outputs.

Setting one of the G inputs to logic 1 effectively turns off the device, causing all Y outputs for that section to take on a logic-1 condition, regardless of the status of the C input.

As clearly shown in the sections that follow, this decoder device plays a vital role in altering the data directed to the matrix-generating schemes already presented in Chapter 4.

Figure 11-2 shows a simple 4-frame animation sequence. The figure in this case is that of a man walking into or out of the plane of



B A	G1	1Y0	1Y1	1Y2	1Y3
X X	1	1	1	1	1
0 0	0	$\overline{C1}$	1	1	1
0 1	0	1	$\overline{C1}$	1	1
1 0	0	1	1	$\overline{C1}$	1
1 1	0	1	1	1	$\overline{C1}$

B A	G2	2Y0	2Y1	2Y2	2Y3
X X	1	1	1	1	1
0 0	0	C2	1	1	1
0 1	0	1	C2	1	1
1 0	0	1	1	C2	1
1 1	0	1	1	1	C2

Fig. 11-1. The 74155 dual 2-line-to-4-line decoder. (a) Logic diagram. (b) Pinout. (c) Truth tables.

the screen. In frame 1, his feet are both on the "ground" and his arms are straight down at his sides. He then takes a step with the left foot in frame 2, extending his right arm and bending up the left one slightly. He returns to the basic position once again in frame 3, and then takes the next step with his right foot in frame 4. Presenting these four frames in sequence and at a rate of about 2 Hz creates the illustration of a walking figure.

Any one of these four figures can be generated by the 8 × 8 extended matrix circuit shown in Fig. 4-17. The trick is to generate all four figures without having to build four separate matrix-generator circuits.

Notice, for instance, that frames 1 and 3 are identical in every respect. It is thus possible to generate both of these frames from the same matrix-generator programming. Frame 2 is obviously different from frames 1 and 3, but it is quite similar to frame 4. Frames 2 and 4, in fact, are simply mirror images of one another, with the mirror axis running vertically through the middle of the matrix. The significance of this mirror effect is that frame 4 can be generated from the same matrix generator for frame 2, but making the horizontal counting take place in the opposite direction.

To express the differences between frames 2 and 4 in another fashion, consider the matrix fold-over scheme described in Chapter 4. In that instance, the size of a symmetrical figure could be doubled in complexity by simply reversing the direction of the matrix addressing half way through the operation. In this case, the direction of addressing for horizontal counts is reversed through the entire frame, frame 4.

The four-frame sequence in Fig. 11-2 can thus be digested down to two basic figures, frames 1 and 2. The programming for the data fed to the matrix generator takes on one particular form when generating frame 1, then the data is changed to generate the image in frame 2. The animation sequence returns to the first set of data programming to generate frame 3. And finally, two things happen at once to generate frame 4: The data programming is returned to that of frame 2 and the direction of horizontal counting is reversed.

So the first step in building an animated video sequence is to plan the frames, keeping them as simple and as much alike as possible. The second step is to analyze the frames, attempting to take advantage of their similarities and reducing the amount of data programming required.

Ultimately the four frames in Fig. 11-2 will be called up in sequence, and since there are four frames, the calling operation

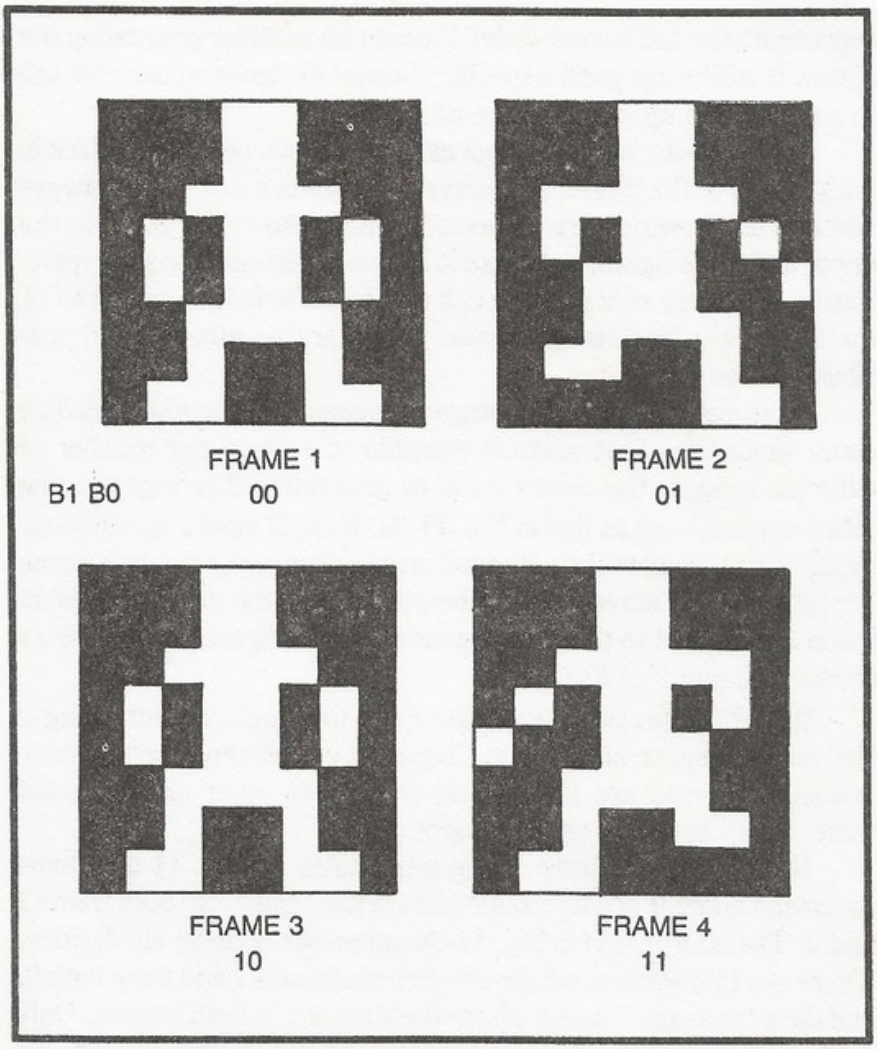


Fig. 11-2. A simple 4-frame animation sequence.

requires two address lines. The frame addresses, designated B1 and B0, determine which one of the four frames will appear on the screen at any given moment. When B1=B0=0, for example, the first frame should appear on the screen. Then when B1=0 and B0=0, the second frame should appear, and so on through the sequence. If this 2-bit counting sequence is taken from a binary counter circuit, the sequencing will take place automatically, thereby generating the desired animation effect.

The table in Fig. 11-3a shows a complete breakdown of frames 1 and 2 in terms of the data inputs required for an 8 × 8 extended matrix generator circuit. The procedure for determining the data requirements is identical to that described in connection with the basic 8 × 8 extended matrix in Fig. 4-18. The "X" in this case

represents the 3rd lowest-order V-count bit used for generating the figure. It will be assigned a specific V-count designation once we are in a position to specify the size of the figure.

In Fig. 11-3a, a 0 entry represents a matrix cell that is black in both halves of the figure. A 1 entry represents a cell that is always white. The X entries are necessary where the cell is black in the upper half of the figure, but white in its lower, extended counterpart. Finally, a $\overline{X}$ entry represents a cell that is white in the upper half of the frame, but black in the lower. There are no other logical possibilities than these.

After making up the sequence of frames and, hopefully, finding some similarities that make it possible to reduce the number of different images, the next step is to generate a figure-generating data sequence such as that in Fig. 11-3a. If the D inputs specified for frame 1 are applied to the extended matrix circuit, the image in frame 1 appears on the screen. If, on the other hand, the data specified in frame 2 is applied to the matrix generator, the figure will appear as shown in frame 2 of Fig. 11-2.

It ought to be clear at this point that a thorough understanding of the complex figure schemes in Chapter 4 are all important to video animation. If you are lost at this point, you must return to the material in Chapter 4 to get caught up.

Now a careful study of the truth table in Fig. 11-3a should uncover a number of data inputs that are the same from both frame 1 and 2. The information in Fig. 11-3b summarizes these similarities. There are 11 instances where the data for frames 1 and 2 are both 0, and then there are 7 cases where the data is 1 in both frames. Only D17 and D25 are totally unique, showing 01 and $1\overline{X}$ respectively. So of the two pairs of 32 data inputs required for this particular animation sequence, there are only 8 different combinations of 0s, 1s, Xs and $\overline{X}$s. Simplification. That is the key to the successful design of video animation circuitry.

The eight different combinations are summarized in Fig. 11-3c. The eight sequences, for the sake of simplicity, are designated according to the first term in the equality expressions in Fig. 11-3b. Bear in mind, for instance, that the programming for D1 in Fig. 11-3c will also apply to data inputs D6 and D26, as specified in Fig. 11-3b.

The next step is to derive some logic circuitry that will alter the data inputs to a single 8 × 8 extended matrix circuit, distinguishing frame 1 data from that required for frame 2 by the status of a control bit, B0. When B0=0, for example, D2 should be equal to logic 0, but

A

D INPUTS	FRAME 1	FRAME 2
0	0	0
1	X	0
2	0	X
3	1	1
4	1	1
5	0	0
6	X	0
7	0	X
8	0	0
9	0	0
10	X	X
11	1	1
12	1	1
13	X	X
14	0	0
15	0	0

D INPUTS	FRAME 1	FRAME 2
16	0	0
17	0	1
18	1	1
19	$\overline{X}$	$\overline{X}$
20	$\overline{X}$	$\overline{X}$
21	1	1
22	0	0
23	0	0
24	0	0
25	1	$\overline{X}$
26	X	0
27	$\overline{X}$	$\overline{X}$
28	$\overline{X}$	$\overline{X}$
29	X	X
30	1	1
31	0	0

B

D0 = D5 = D8 = D9 = D14 = D15 = D16 = D22 = D23 = D24 = D31 = 0
D1 = D6 = D26
D2 = D7
D3 = D4 = D11 = D12 = D18 = D21 = D30 = 1
D10 = D13 = D29
D17
D19 = D20 = D27 = D28
D25

C

	B0	D0	D1	D2	D3	D10	D17	D19	D25
FRAME 1	0	0	X	0	1	X	0	$\overline{X}$	1
FRAME 2	1	0	0	X	1	X	1	$\overline{X}$	$\overline{X}$

Fig. 11-3. Truth-table analysis of the 4-frame animation sequence in Fig. 11-2. (a) D-input programming for frames 1 and 2. (b) Equations showing data inputs having identical programming for both primary frames. (c) simplified truth table for matrix D-input programming.

when B0 is changed to logic 1, input D2 to the matrix generator should be changed to X.

Making this step calls for a rather experienced outlook on digital logic design. The procedures are rather straight-forward for experi-

enced technicians, but a beginner will find the task a rather troublesome one at times.

What is the result of all this analysis and simplification? The answer is contained in the fairly simple circuit in Fig. 11-4. Here we have the complete circuit for generating the 4-frame animated sequence in Fig. 11-2, and also a prime example of how it is possible to devise some simple circuitry for carrying out what appears at first to be an exceedingly difficult task. In this particular example, a 4-frame animated sequence is simplified to a point where it can be implemented with an 8 × 8 extended matrix generator and only four outboard IC devices.

Comparing the tables in Fig. 11-3 with the circuit in Fig. 11-4, it can be seen that the data inputs requiring a logic 0 in all cases can be connected together to COMM, while those calling for a constant logic-1 level can be connected together at +5V. These two simple operations take care of 18 of the D inputs for both frames.

Furthermore, the D10 input is always equal to X and D19 is at the $\overline{X}$ level. The X input in Fig. 11-4 is thus connected directly to D10, while D19 sees a version of the same input that is first inverted by IC3-B.

All that remains as far as the D inputs are concerned are those designated D1, D2, D7, and D25 in Fig. 11-3c. This set of inputs differs from the others inasmuch as they vary according to the frame being displayed. Input D1, for instance, is equal to X during frame 1 (B0=0), but must be set at logic 0 through frame 2 (B0=1).

The B0 signal—the one distinguishing data for frame 1 from that of frame 2—is generated by the least-significant output of a 2-bit binary counter, IC2-A and IC2-B. Whenever B0 is equal to 0, IC3-A is effectively gate on, and an inverted version of the $\overline{X}$ signal from IC3-B is presented to D1. D1, in other words, is set to X whenever B0=0 (a condition that satisfies the requirements established for D1 in Fig. 11-3c). Whenever B0 switches to logic 1, however, it gates off IC3-A and guarantees a logic-0 level at D1.

The data for D2 is derived in a similar fashion, taking its $\overline{X}$ data from IC3-B and frame-select data from the $\overline{B0}$ output of IC2-A. When B0=0, then, $\overline{B0}$ is equal to logic 1, and IC3 is effectively gated off to feed a guaranteed logic-0 level to D2. Whenever B0 switches to logic 1, however, $\overline{B0}$ is set to logic 0, and D2 sees an X signal. These conditions meet the requirements spelled out for D2 in Fig. 11-3c.

According to Fig. 11-3c, D25 is simply an inverted version of the D2 input. That is, D25=1 when D2=0, and D25=$\overline{X}$ when D2=X. All that is necessary for generating the D25 input, then, is an

inverting operation between D2 and D25, and that is performed by IC3-D.

Finally, it can be seen from Fig. 11-3c that D17 is equal to B0. So the final D connection is one where the B0 output of IC2-A is connected directly to the D17 input of the matrix generator.

The circuit as described to this point is capable of distinguishing frames 1 and 2. What remains is the procedure for inverting the direction of horizontal counting in order to produce the mirror images for frames 3 and 4. This is accomplished by inverting the H-count inputs to select lines S0, S1, and S2 while B1=1. And it is a simple matter of running these H-count levels through EXCLUSIVE OR gates, IC4-A through IC4-C. Whether these select lines count forward or in reverse depend on the logic level from IC2-B, counting forward while B1=0 (frames 1 and 2) and counting in reverse while B1=1 (frames 2 and 3).

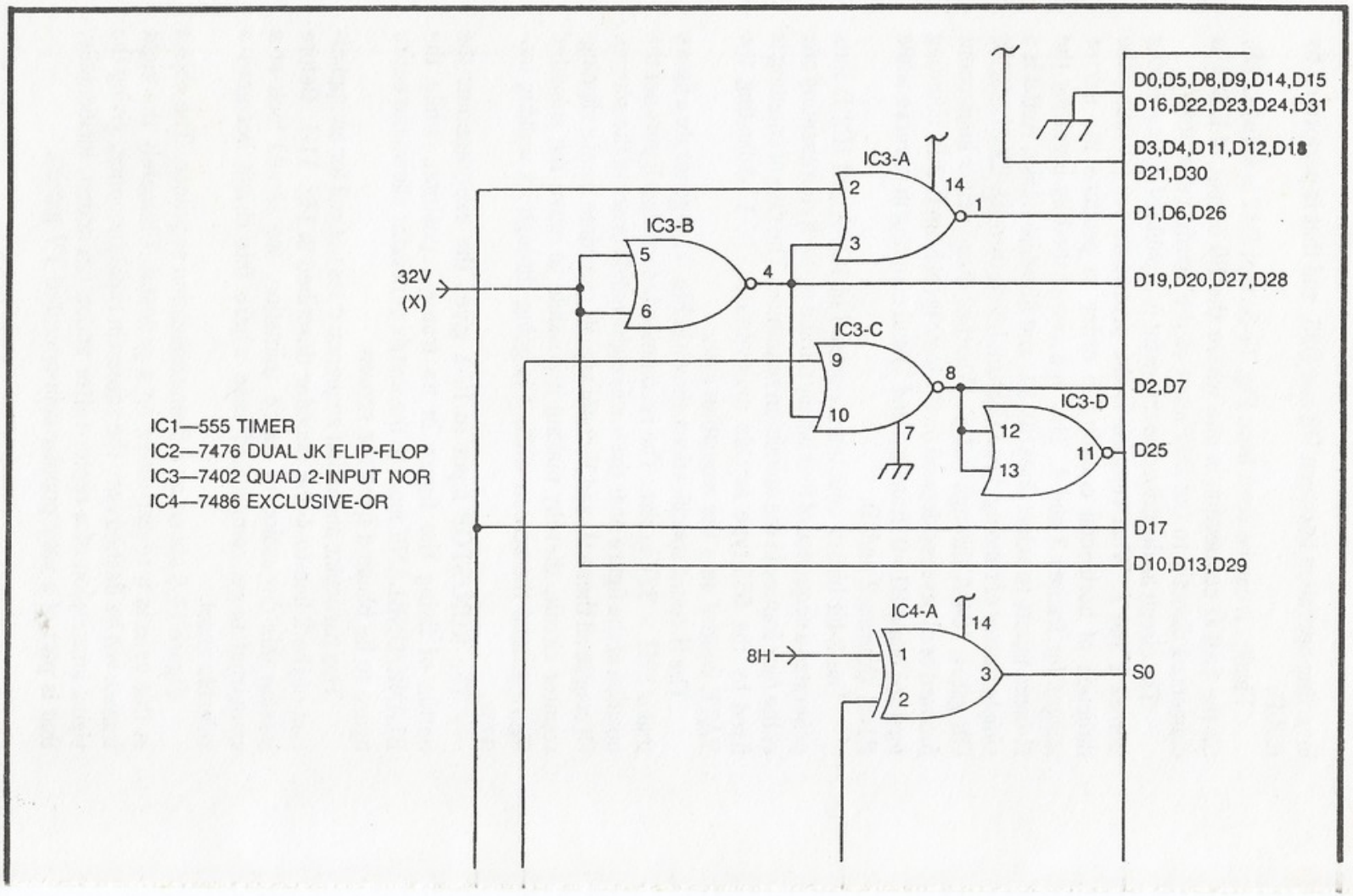
The 2-bit binary counter composed of IC2-A and IC2-B thus generates a sequence of four binary numbers, each representing one of the four frames in the animation sequence. The rate of counting is fixed by the 555-type astable multivibrator, IC-1. Adjusting the RATE control sets the animation rate.

The H-count specifications shown in Fig. 11-4 generate a figure that is 32H × 32V in size. The recommended window inputs set the position of the figure near the lower right-hand corner of the screen. Of course all these H- and V-count signals can come from a slipping-counter circuit, thereby making it possible to move the animated figure around the screen while it is going through its walking motions.

The RUN/STOP input to IC-1 gives the experimenter the option of fixing the figure in its frame-1 position, while the BLANK/UNBLANK input to the matrix generator allows the entire figure to be blanked from the screen.

This particular animation sequence does not call for an application of the 2-line-to-4-line decoder described in Fig. 11-1. Before leaving this discussion of figure animation, we should look at a somewhat more complicated case where this circuit becomes a valuable asset.

Figure 11-5 shows an 8-frame animation sequence. The object in this case is a teeter-totter on a pedestal. Ultimately the eight frames will be flashed onto the screen in rapid sequence, giving the visual impression of a teeter-totter action (an action, incidentally, that is part of a very popular coin-operated TV game).


IC1—555 TIMER
IC2—7476 DUAL JK FLIP-FLOP
IC3—7402 QUAD 2-INPUT NOR
IC4—7486 EXCLUSIVE-OR
32V
(X)
IC3-A
IC3-B
IC3-C
IC3-D
IC4-A
8H
D0,D5,D8,D9,D14,D15
D16,D22,D23,D24,D31
D3,D4,D11,D12,D18
D21,D30
D1,D6,D26
D19,D20,D27,D28
D2,D7
D25
D17
D10,D13,D29
S0

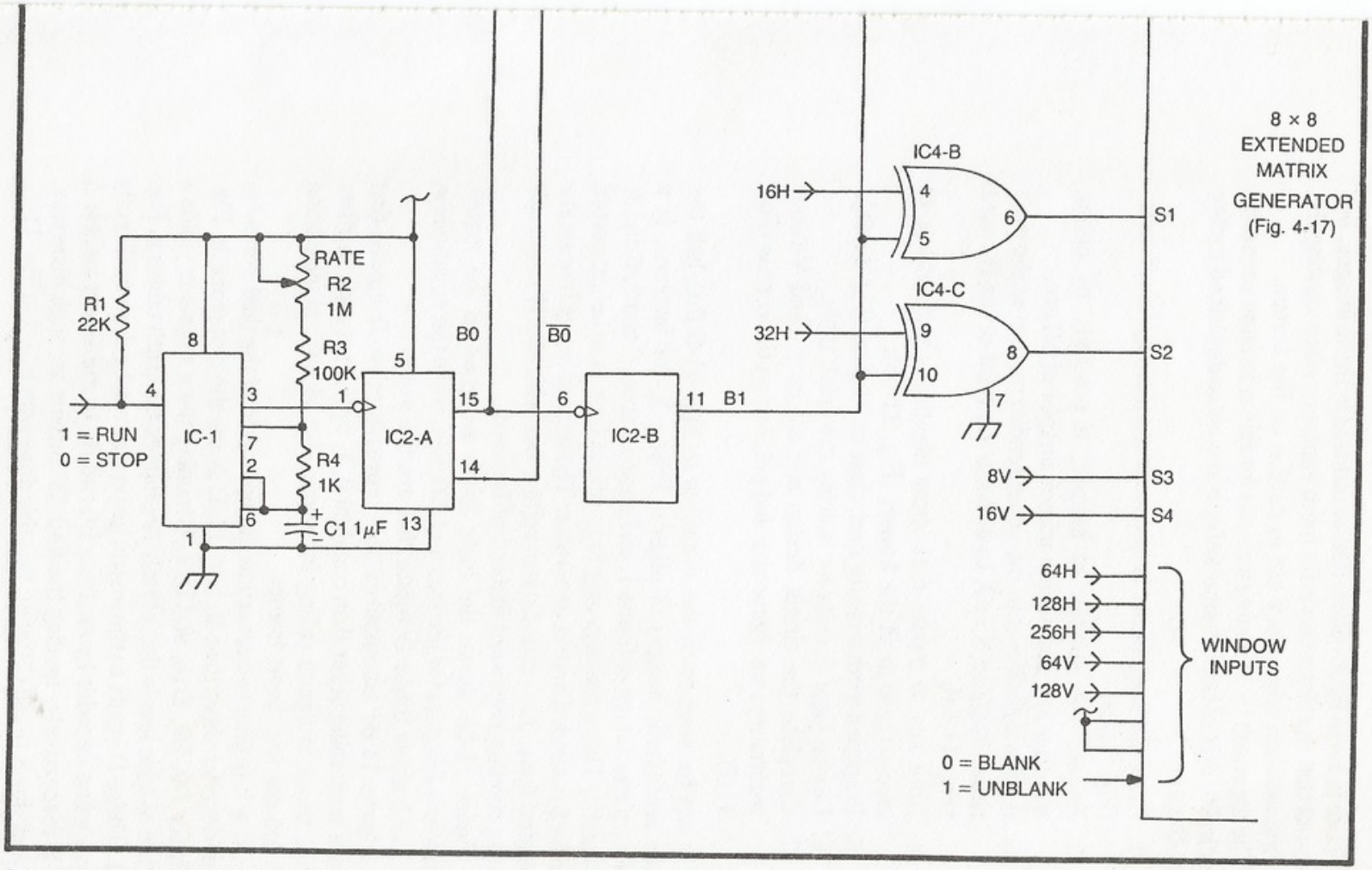


Fig. 11-4. Circuit for producing the 4-frame animation sequence in Fig. 11-2.

There is some figure distortion introduced in these images, but it turns out that the distortion is far more apparent when viewing the static frames than when they are in motion on the screen.

The approach to designing a circuit for this animation sequence is identical to that of the 4-frame walking sequence described earlier in this chapter:

1. Draw the basic figure sequence in standard matrix formats (Fig. 11-5).
2. Reduce the number of images, if possible, by noting whether any are simple mirror images of others.
3. Assign logic levels to the nonextended portion of the matrices, using an X or $\overline{X}$ to indicate the third-order H-count (Fig. 11-6a).
4. Note any D inputs that show identical combinations of inputs through all the frames (Fig. 11-6b).
5. Prepare a truth table for each class of D inputs (Fig. 11-6c).
6. Devise logic circuits for solving the truth table.
7. Complete the circuit design and assign H- and V-count parameters for figure size and position on the screen (Fig. 11-7).

It can be seen from the drawings in Fig. 11-5 that half the figures are mirror images of others. Frame 8, for instance, is a vertical mirror image of frame 1, and so are frames 7 and 2, 6 and 3, and 5 and 4. There are thus only four unique frames to be generated by an 8×8 extended matrix generator. The mirror-image frames are generated from the same data circuitry, but with the H-count addresses running backward instead of forward.

Figure 11-6a shows the logic levels assigned to the matrix generator's D inputs for the four basic frames, while the equations in Fig. 11-6b show those D inputs that are the same.

Figure 11-6c summarizes the representative D inputs that require something other than constant 0, 1, or X. It shows, in other words, those D inputs calling for some sort of logic manipulation between the four basic frames.

Now it is time to appreciate the operation of the dual 2-line-to-4-line decoder described in Fig. 11-1. Note the sequence in Fig. 11-6c for D0, D8, D16, and D24. Grouping them together yields a pattern of logic levels that closely resembles the truth tables in Fig. 11-1. These D inputs to the matrix generator can be derived directly from a 4-line decoder by applying B0 and B1 to the select inputs, A and B respectively, feeding the four 2Y outputs through inverters, and applying X to the C2 input. See this done with IC3-A in Fig. 11-7.

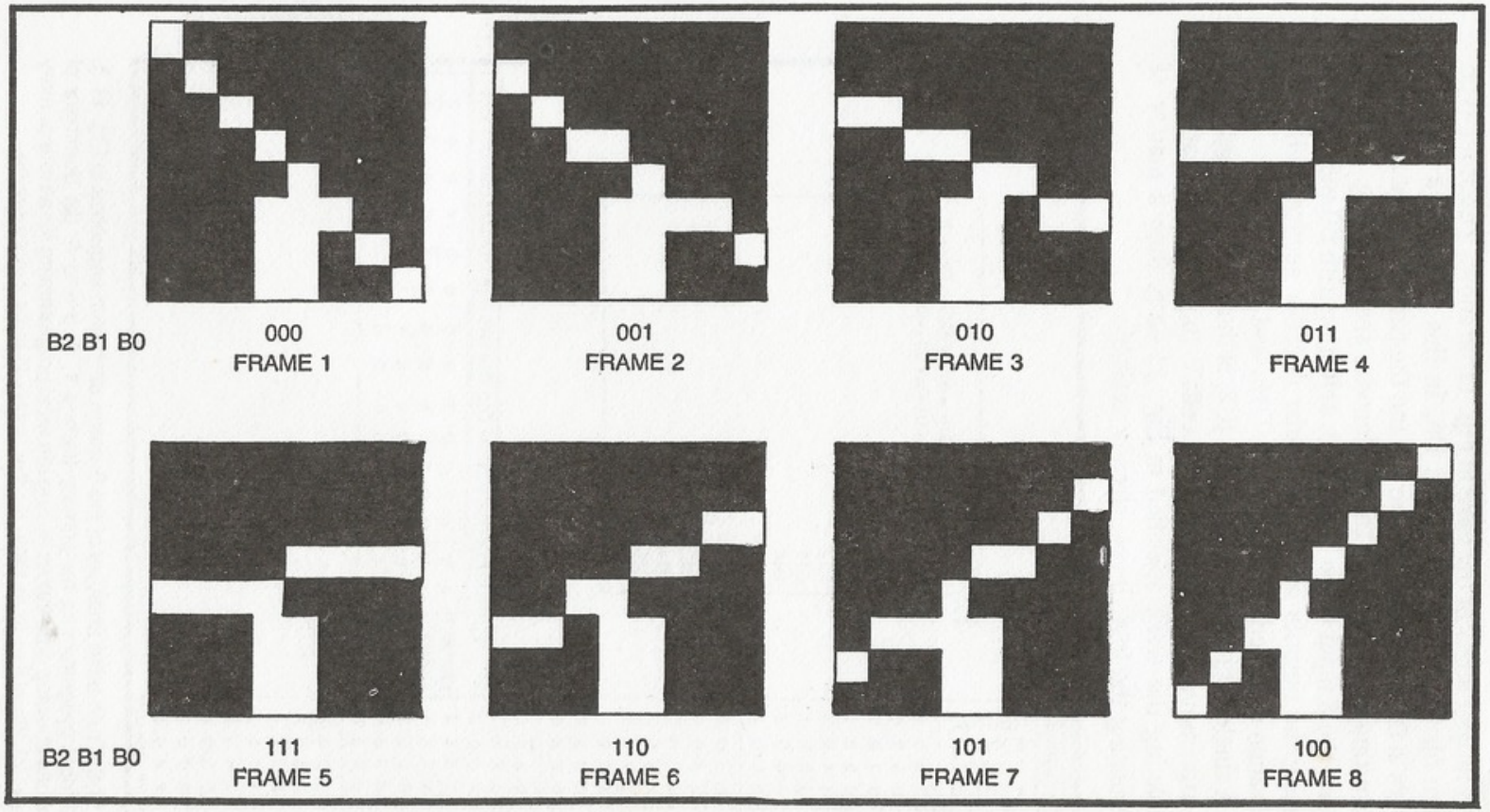


Fig. 11-5. An 8-frame animation sequence that gives the visual impression of a teeter-totter.

A further study of the table in Fig. 11-6c shows a decoder-type pattern for D22, D23, D15, and D6. In this instance, a single X appears in the company of three 0s, and the pattern is that achieved by inverting the outputs of the 1Y decoder section in Fig. 11-1. So applying the X input to pin 1 of IC3-B, selecting with B0 and B1, and inverting the outputs of that particular decoder section yields the information required for D22, D23, D15, and D6.

A single dual-decoder package, IC3, is thus capable of generating programmable data inputs for 12 cells in the animation matrices. Completing the work specified in Fig. 11-6c is then a matter of manipulating the logic levels already available.

A

D	POSITION FRAME 1	FRAME 2	FRAME 3	FRAME 4
0	$\overline{X}$	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	X	X	X	X
5	0	0	X	X
6	0	0	0	X
7	0	0	0	X
8	0	$\overline{X}$	0	0
9	$\overline{X}$	0	0	0
10	0	0	0	0
11	X	X	X	X
12	X	X	X	X
13	X	X	0	0
14	0	X	X	0
15	0	0	X	0
16	0	0	$\overline{X}$	0
17	0	$\overline{X}$	$\overline{X}$	0
18	X	0	0	0
19	$\overline{X}$	X	X	X
20	X	X	X	X
21	0	0	0	0
22	X	0	0	0
23	0	X	0	0
24	0	0	0	$\overline{X}$
25	0	0	0	$\overline{X}$
26	0	$\overline{X}$	$\overline{X}$	$\overline{X}$
27	1	1	1	1
28	X	X	X	X
29	0	0	0	0
30	0	0	0	0
31	X	0	0	0

B

D0 = D9 = D18
D1 = D2 = D3 = D10 = D21 = D29 = D30 = 0
D4 = D11 = D12 = D19 = D20 = D28 = X
D5
D6 = D7
D8
D13
D14
D15
D16
D17
D22 = D31
D23
D24 = D25
D26
D27 = 1

C

	B1	B0	D0	D5	D6	D8	D13	D14	D15	D16	D17	D22	D23	D24	D26
FRAME 1	0	0	$\overline{X}$	0	0	0	X	0	0	0	0	X	0	0	0
FRAME 2	0	1	0	0	0	$\overline{X}$	X	X	0	0	$\overline{X}$	0	X	0	$\overline{X}$
FRAME 3	1	0	0	X	0	0	0	X	X	$\overline{X}$	$\overline{X}$	0	0	0	$\overline{X}$
FRAME 4	1	1	0	X	X	0	0	0	0	0	0	0	0	$\overline{X}$	$\overline{X}$

Fig. 11-6. Truth-table analysis of the 8-frame animation sequence in Fig. 11-5. (a) D-input programming for primary frames 1, 2, 3 and 4. (b) Summary of equations showing data inputs having identical programming for all four primary frames. (c) Simplified truth table for the matrix D-input programming.

Input D17, for example, uses two $\overline{X}$ terms that can be derived by ORing together D8 and D16. IC6-A accomplishes this task, taking advantage of the fact that inverted data at the input of a NAND gate yields an OR function. D14 is obtained in a similar fashion, effectively ORing together D15 and D23. IC7-A and IC4-E work together to OR D17 and D24 to produce the D26 input.

D5 and D13 could be derived from ORed combinations of other D inputs, but more for the sake of illustration than anything else, the circuit in Fig. 11-7 shows them being generated in a different manner. D5 seems to have a close relationship with the B1 frame-select bit (D5=0 when B1=0, and D5=X when B1=1). So it is possible to generate the D5 signal by gating it off while B1=0 and allowing X to emerge when B1=1. This is a simple AND operation performed by the NOR gate, IC7-7, operating from inverted versions of B1 and X.

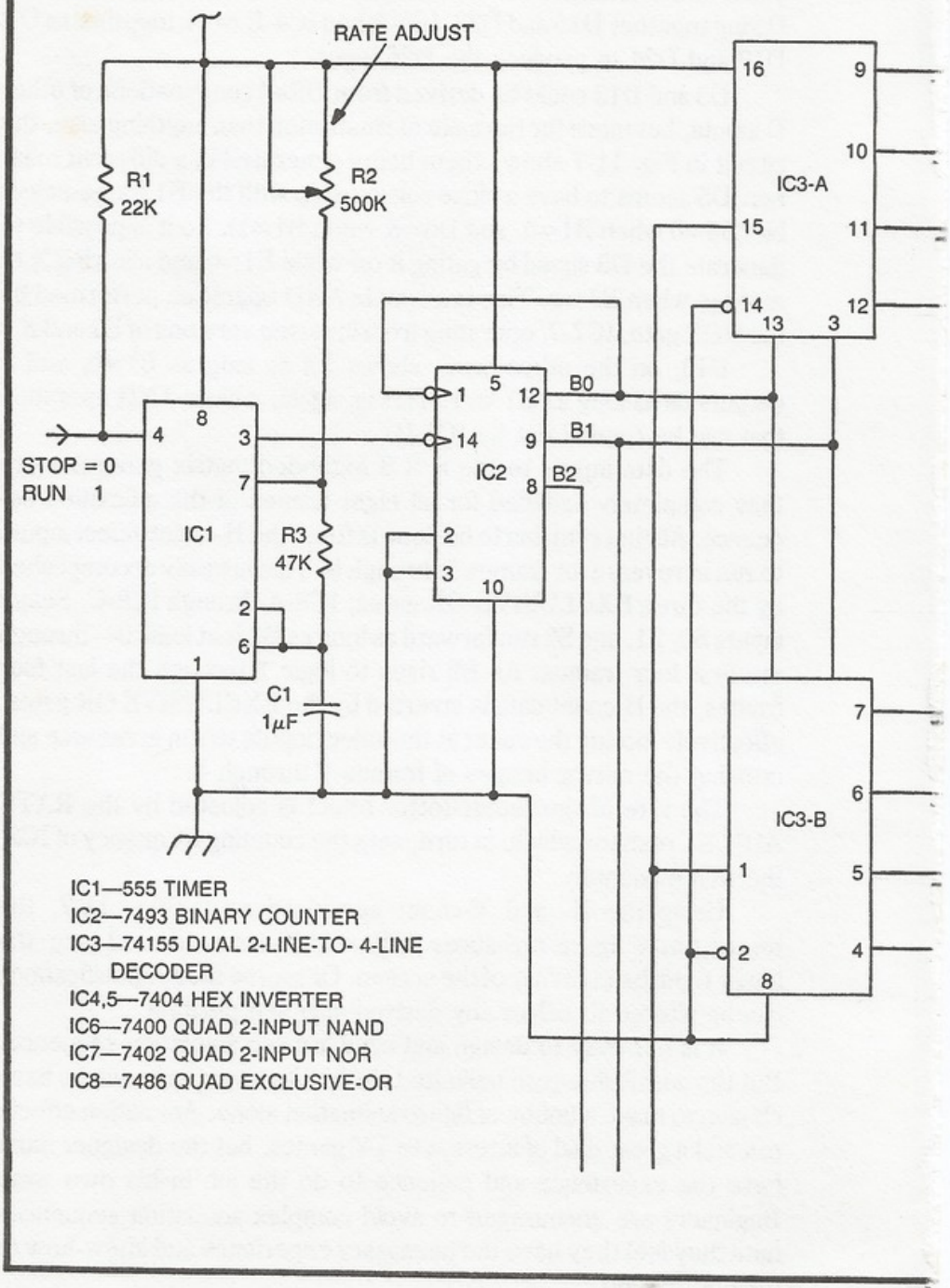
D13, on the other hand, shows Xs as long as B1=0, and it outputs 0s as long as B1 = 1. This is, again, a basic AND operation that can be carried out by IC7-B.

The data inputs to the 8 × 8 extended matrix generator are thus completely satisfied for all eight frames of the animation sequence. All that remains to be done is force the H-count select inputs to run in reverse for frames 5 through 8. This is easily accomplished by the three EXCLUSIVE OR gates, IC8-A through IC8-C. Select inputs S0, S1, and S2 run forward as long as B2 is at logic 0—through the first four frames. As B2 rises to logic 1 through the last four frames, the H-count data is inverted by the EXCLUSIVE OR gates, effectively forcing the count at the select inputs to run in reverse and creating the mirror images of frames 1 through 4.

The rate of the teeter-totter effect is adjusted by the RATE ADJUST resistor which, in turn, sets the counting frequency of IC2, the frame counter.

Using the H- and V-count specifications in Fig. 11-7, the teeter-totter figure measures 32H × 32V and is located near the lower right-hand corner of the screen. Of course these specifications can be altered to select any desired size and position.

It is not easy to design and build a video animation sequence. But the possibilities are unlimited. In fact some experimenters have chosen to make a hobby of figure animation alone. Animation effects can add a great deal of interest to TV games, but the designer must have the experience and patience to do the job in his own way. Beginners are encouraged to avoid complex animation sequences until they feel they have the necessary experience and know-how to take on the job.


RATE ADJUST
R1
22K
R2
500K
R3
47K
C1
1μF
IC1
IC2
IC3-A
IC3-B
STOP = 0
RUN 1
B0
B1
B2
IC1—555 TIMER
IC2—7493 BINARY COUNTER
IC3—74155 DUAL 2-LINE-TO- 4-LINE DECODER
IC4,5—7404 HEX INVERTER
IC6—7400 QUAD 2-INPUT NAND
IC7—7402 QUAD 2-INPUT NOR
IC8—7486 QUAD EXCLUSIVE-OR

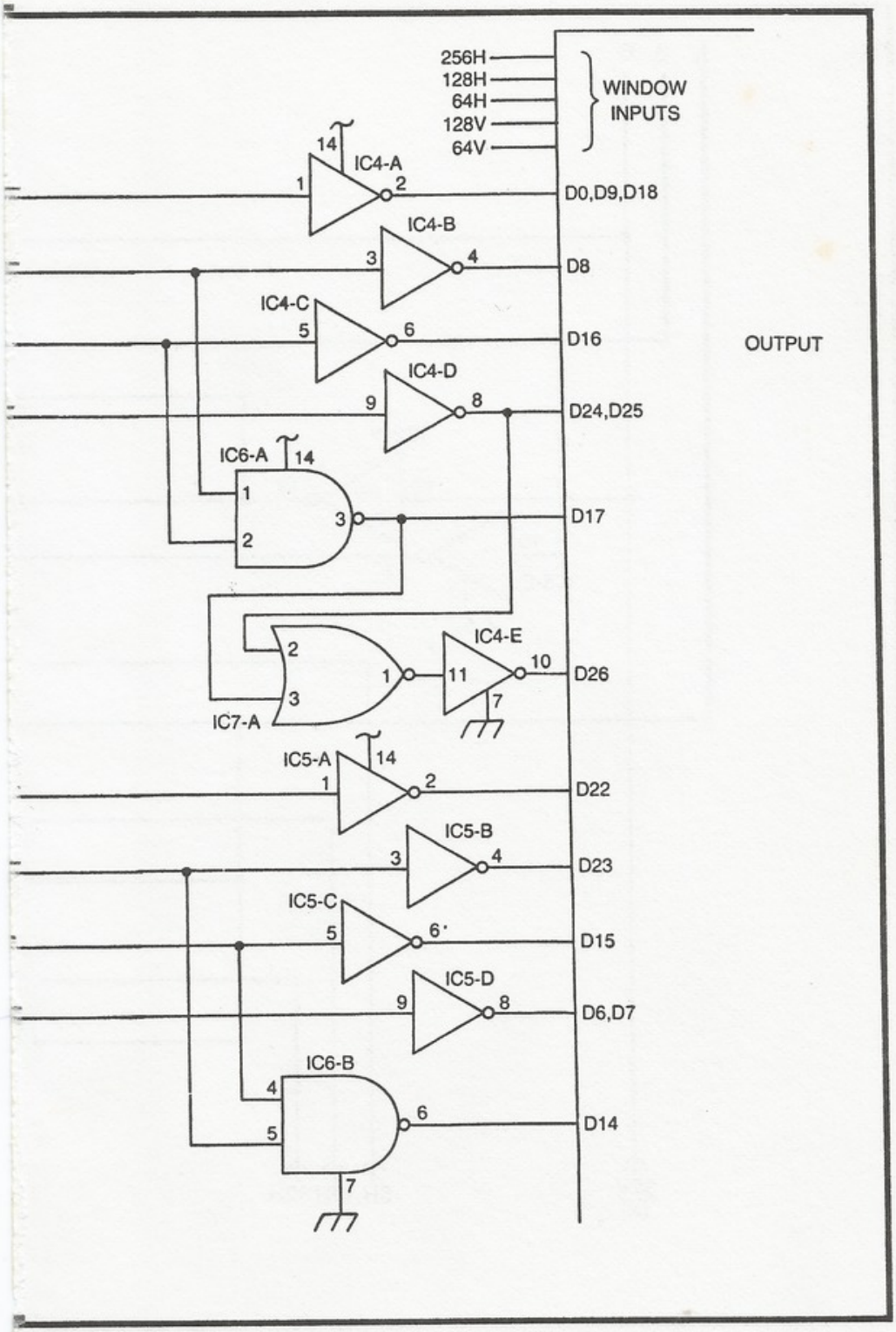


Fig. 11-7. Circuit for producing the 8-frame animation sequence in Fig. 11-5

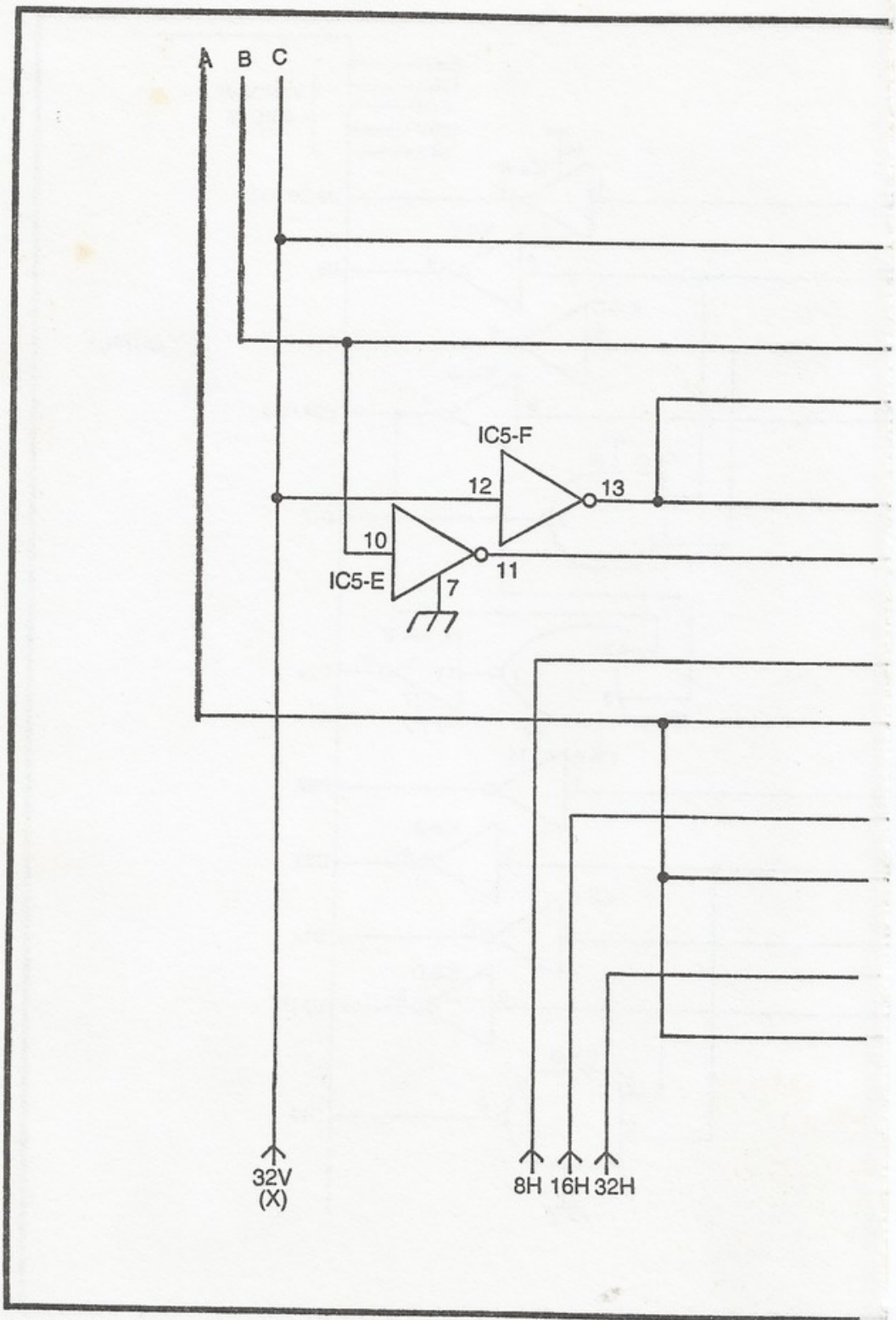


Fig. 11-7. Continued.

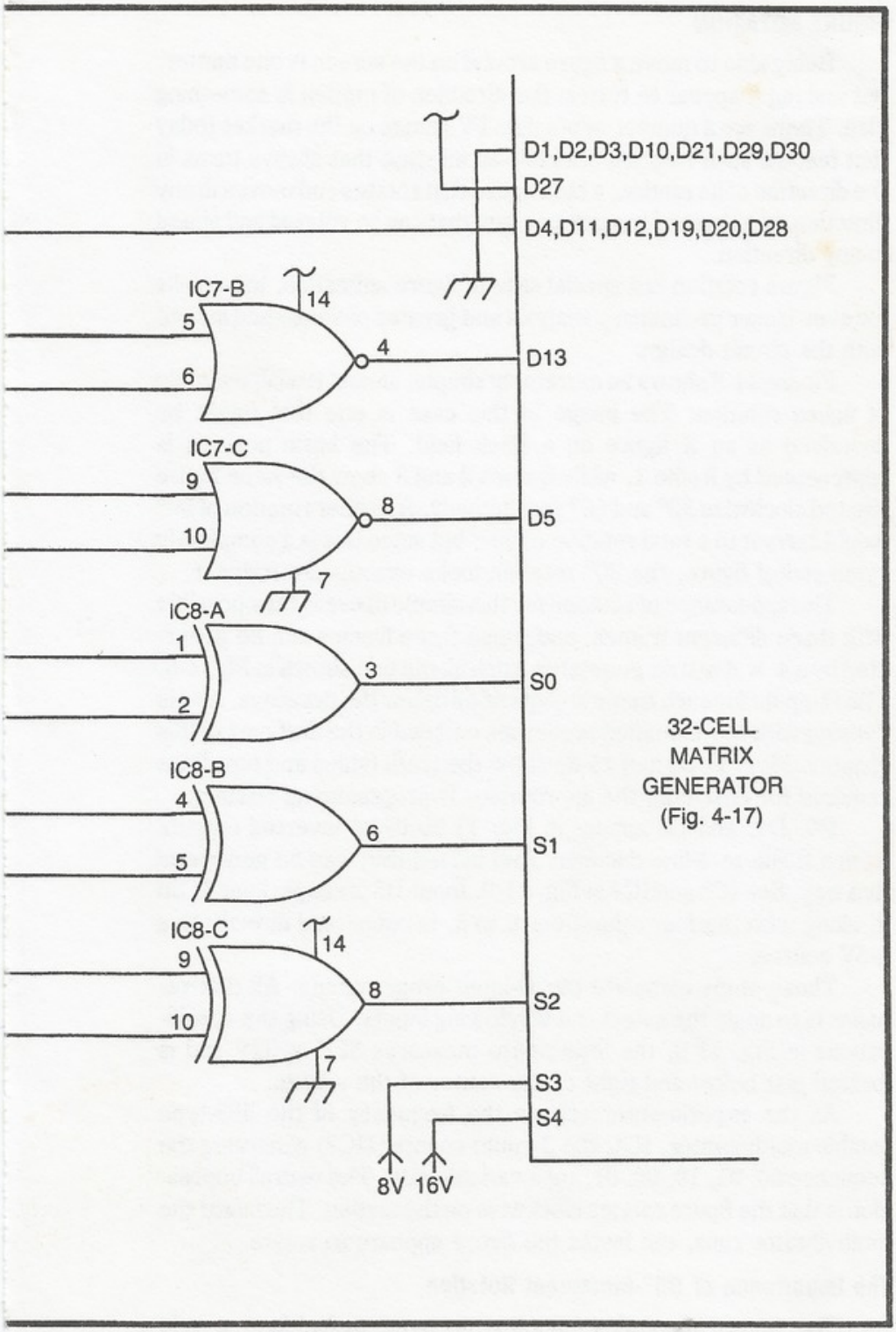


Fig. 11-7 Continued.

FIGURE ROTATION

Being able to move a figure around on the screen is one matter, but making it appear to turn in the direction of motion is something else. There are a number of popular TV games on the market today that feature such rotation effects—an airplane that always turns in the direction of its motion; a tank figure that rotates and moves in any direction, but always forward; or a gun that can be rotated and aimed in any direction.

Figure rotation is a special case of figure animation, but it calls for even closer preliminary analysis and greater patience and insight with the circuit design.

Figure 11-8 shows an extremely simple, almost trivial, example of figure rotation. The image in this case is one that might be described as an X figure on a black field. The basic position is represented by frame 1, while frames 2 and 3 show the same figure rotated clockwise 30° and 60° respectively. A further rotation of 30° would carry it to a total rotation of 90°; but since this is a completely symmetrical figure, the 90° rotation looks exactly like frame 1.

The appearance of rotation for this simple figure is thus possible with three different frames, and those three frames can be generated by a 4 × 4 matrix generator, such as the one shown in Fig. 4-5. The D inputs for each frame are specified below the drawings. And in keeping with the animation processes outlined in the first part of this chapter, Figs. 11-8b and 11-8c show the truth tables and equalities required for designing the appropriate D-programming circuitry.

D0, D1, and D2 appear in Fig. 11-8b to be inverted outputs from a 2-line-to-4-line decoder. And indeed they can be generated that way. See IC3 and IC4 in Fig. 11-9. Input D5 is simply logic 1. So it, along with the four others equal to it, is connected directly to a +5V source.

These steps complete the D-input programming. All that remains is to apply the select and windowing inputs. Using the specifications in Fig. 11-9, the little figure measures 32H × 32V and is located just below and right of the center of the screen.

As the experimenter adjusts the frequency of the 555-type astable multivibrator, IC1, the 3-count counter (IC2) generates the sequence 00, 01, 10, 00, 01...at a variable rate. The overall impression is that the figure rotates clockwise on the screen. The faster the multivibrator runs, the faster the figure appears to rotate.

The Importance of 90°-Increment Rotation

The matter of rotating complex, nonsymmetrical figures calls for some techniques that aren't required for the simpler forms of

A

B1B0	FRAME 1 00	FRAME 2 01	FRAME 3 10
D0	1	0	0
D1	0	1	0
D2	0	0	1
D3	1	0	0
D4	0	0	1
D5	1	1	1
D6	1	1	1
D7	0	1	0
D8	0	1	0
D9	1	1	1
D10	1	1	1
D11	0	0	1
D12	1	0	0
D13	0	0	1
D14	0	1	0
D15	1	0	0

B

B1 B0	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
0 0	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
0 1	0	1	0	0	0	1	1	1	1	1	1	0	0	0	1	0
1 0	0	0	1	0	1	1	1	0	0	1	1	1	0	1	0	0

C

D0 = D3 = D12 = D15
D1 = D7 = D14
D2 = D4 = D11 = D13
D5 = D6 = D9 = D10

Fig. 11-8. A very simple animation/rotation sequence. (a) Figures and corresponding D-input matrix programming. (b) Simplified truth table. (c) List of D inputs having identical programming through all three frames.

figure animation. To get an appreciation for the situation, suppose you want to rotate a figure through a full 360° turn at increments of 45°. Now that is a very coarse rotation sequence. The figure will appear to jump from one 45° angle to the next in a very unrealistic fashion. But this is simply an example.

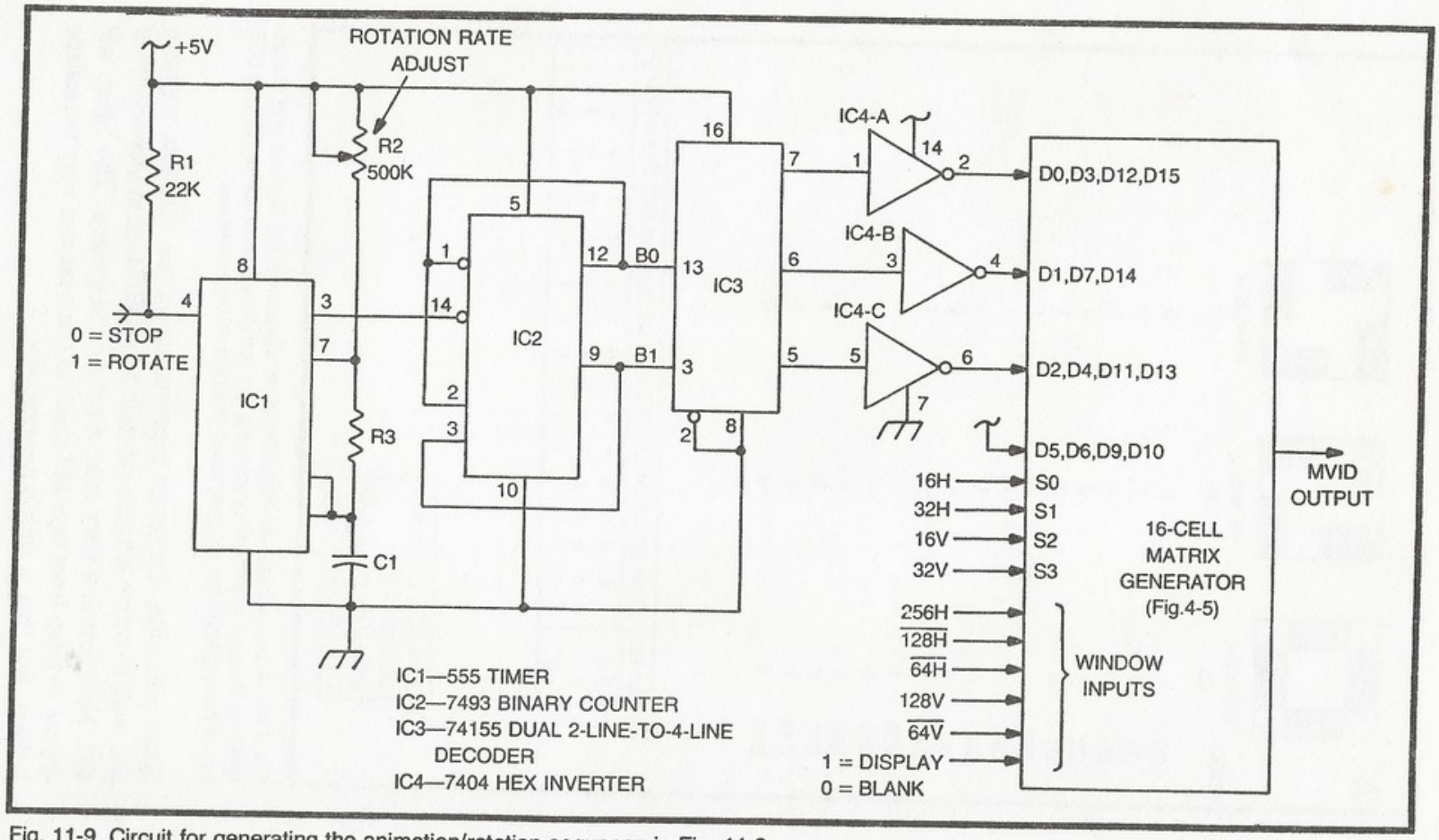


Fig. 11-9. Circuit for generating the animation/rotation sequence in Fig. 11-8.

Now if the figure is going to rotate in 45° steps, it follows that the sequence will use eight animation frames: 0°, 45°, 90°, 135°, 180°, 225°, and 315°. An additional 45° rotation carries the figure back to its original 0° position. See the sequence in Fig. 11-10.

From the foregoing discussion of animation sequences, it might be assumed that one doesn't need eight different sets of D-input matrix programs to accomplish this rotation effect. The 180° figure, for instance, is a vertical reflection of the 0° figure, the 270° figure is a horizontal reflection of the 90° one, the 135° figure is a vertical reflection of the 45° version, and finally the 315° image can be derived by a vertical reflection of the 225° figure. The eight figures can thus be obtained by performing either a horizontal- or vertical-reflection operation on just four of them. It would be possible to generate this sequence by establishing a D-input matrix program for four of them, then reversing the direction of horizontal or vertical counting to get the other four.

That's not too bad. But notice how coarse the rotation effect would be. The player would not see anything resembling a smooth rotation. The figure would appear to snap around in 45° intervals. And overcoming that problem is a matter of rotating the figure in finer angular increments.

So let's suppose you try rotating a complex figure in 22½° increments, cutting the minimum angular increment in half. Does this double the number of images in the animation/rotation sequence? No, it quadruples the number of figures, apparently making

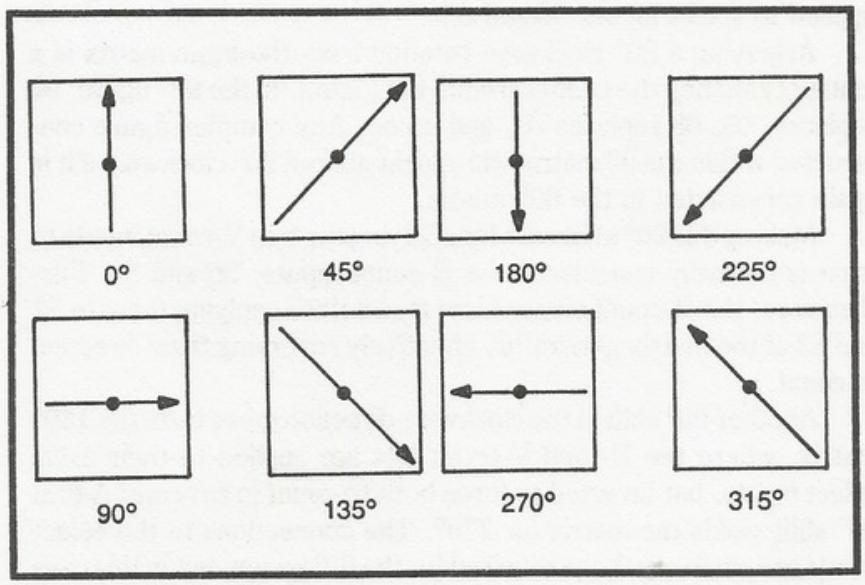


Fig. 11-10. Definitions of rotation through 360 degrees at 45-degree increments.

the whole affair four times as complicated! If anything useful is to be done with this matter of rotating complex figures in a realistic fashion, there has to be a special trick for simplifying the whole thing. That trick is 90° rotations.

The previous analysis of the rotation sequence in Fig. 11-10 showed that it could be done with four programmed figures (0°, 90°, 45°, and 225°) by reversing the horizontal or vertical counting to obtain the other four figures. Using a 90°-shift operation, however, it is possible to generate all eight figures from just two matrix programs, the 0° and 45° images.

While the sequence in Fig. 11-10 is built around 45° increments, it can be seen that the 0° image can be shifted by 90° to get the images for 90, 180, and 270 degrees, while the 45° image can be shifted in 90° increments to get the images for 135, 225 and 315 degrees. Using this 90°-shift technique, it is possible to get all eight frames from the D-input matrix programs for 0° and 45°. And in fact a 16-frame rotation sequence (22½° intervals) can be derived from just four D-input matrix programs.

Figure 11-11 shows the rationale behind rotating a single 4 × 4 matrix image in 90° increments by varying the select format, rather than the D-input programming. In this case, the basic 0° matrix is generated in the usual fashion, applying the least-significant H-count bit (H0) to S0 of the matrix generator, the most-signifcant H-count (H1) to S1, and so on. The numerals in the matrix cells indicate their relative positions, with the first digit being the decimal value of select inputs S0 and S1, and the second being the decimal value of the bits applied to select inputs S2 and S3.

Achieving a 90° clockwise rotation from the basic matrix is a matter of shifting the counts around the matrix. In the 90° figure, 00 replaces, 03, 03 replaces 33, and so on. Any complex figure constructed within the 0° matrix will appear shifted 90° clockwise if it is again constructed in the 90° matrix.

Making this 90° shift calls for applying the two V-count inputs to what is normally considered the H-count inputs, S0 and S1. Furthermore, the H-count bits are inverted before applying them to S2 and S3 of the matrix generator, effectively reversing their direction of count.

Another 90° shift in the clockwise direction takes us to the 180° matrix, where the H- and V-count bits are applied to their usual select inputs, but inverted to force both to count in reverse. A final 90° shift yields the matrix for 270°. The connections to the select inputs are similar to those required for the 90° image, but in this case it is the V-count inputs that run in reverse.

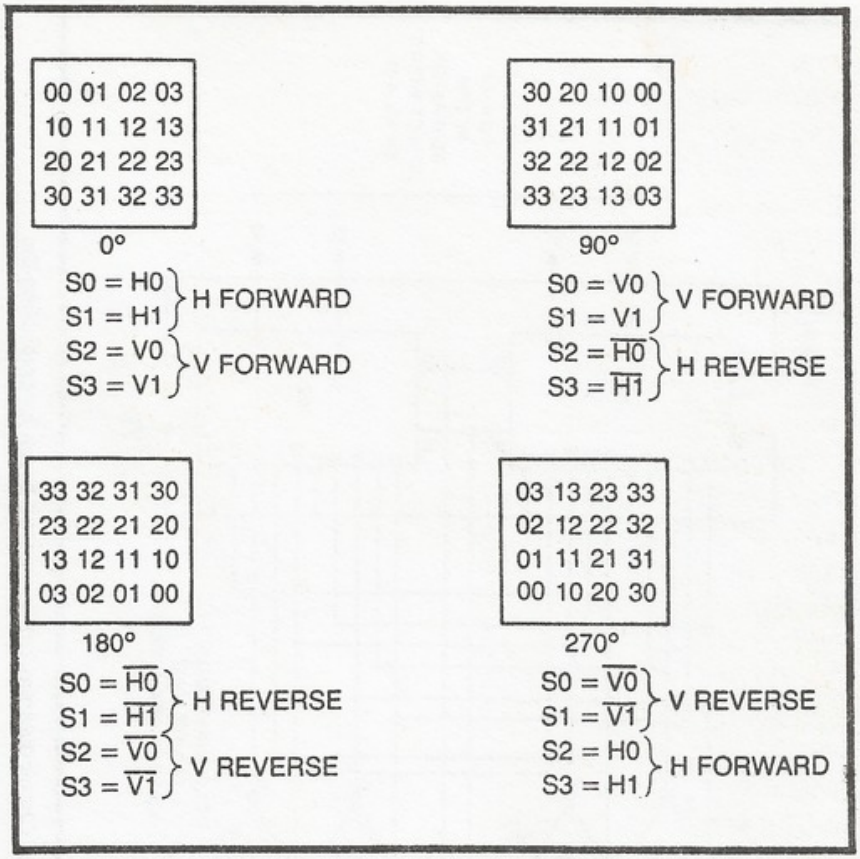


Fig. 11-11. 90-degree shifting matrices and corresponding address specifications.

What we have here is the ability to rotate any 4 × 4 matrix-generated figure in increments of 90°. The table in Fig. 11-12a shows the combinations of H- and V-counts to be applied to the matrix-select inputs at the four different angles. The angles are selected by an external source of 2-bit binary numbers, B0 and B1. When B0=B1=0, for example, the figure will be in its basic 0° position. But when B0=1 and B1=0, the 90° image is selected.

The circuit in Fig. 11-12b shows how the various angle patterns can be obtained by running the two B inputs through their 2-bit binary counting sequence. It is really just a set of four 4:1 multiplexer circuits, each one providing an output for each of the four select lines to the matrix generator.

A careful study of the circuit in Fig. 11-12b shows that it satisfies the truth table in Fig. 11-12a. And since that truth table is derived from the matrix-rotation scheme in Fig. 11-11, it ultimately follows that the circuit in Fig. 11-12b will do the job. A single figure

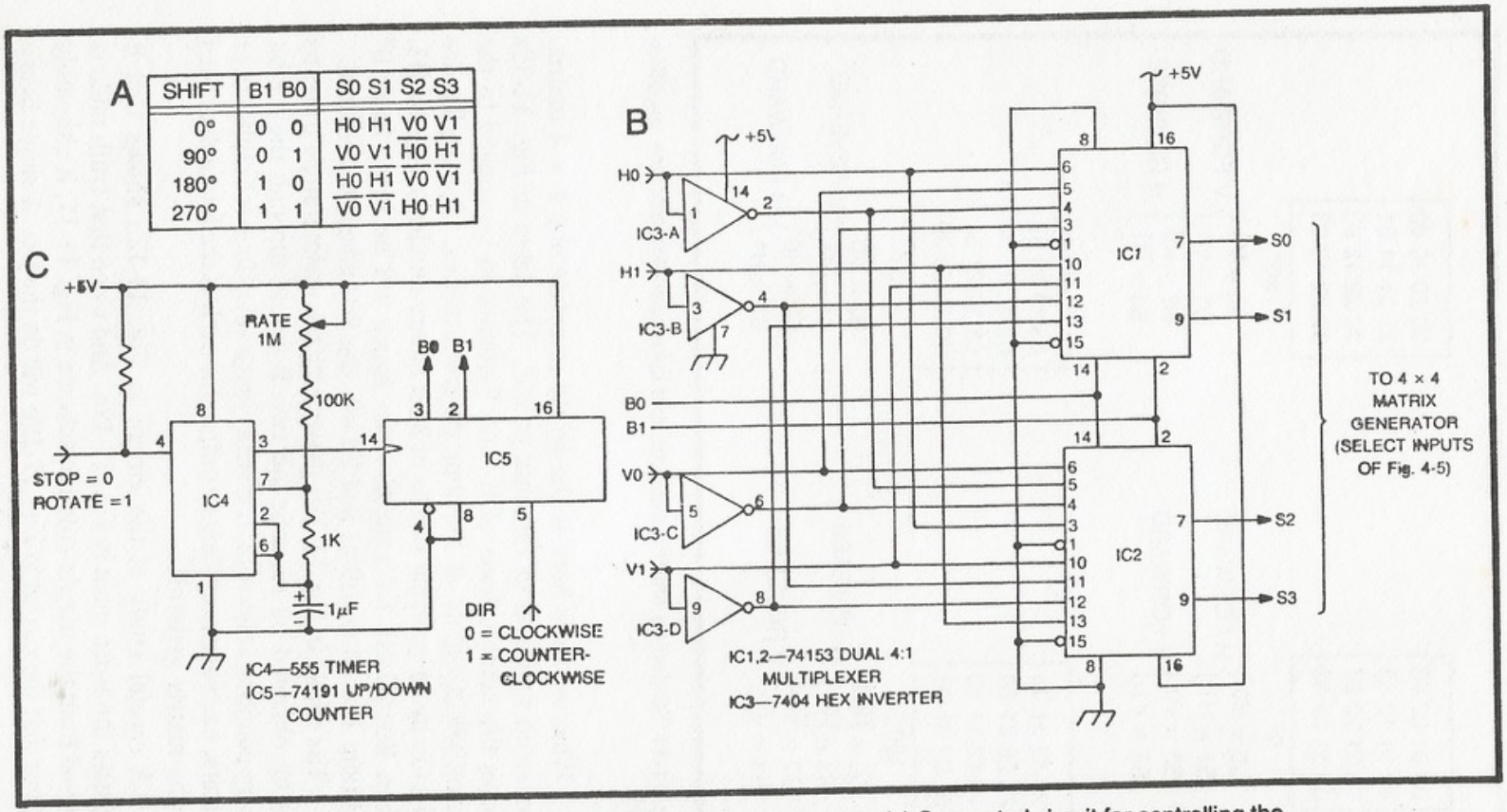


Fig. 11-12. A standard 90° -shift circuit. (a) Truth table rationale. (b) Circuit diagram. (c) Suggested circuit for controlling the rate and direction of rotation.

programmed into the D inputs of the matrix generator can be rotated to any one of four different positions.

The circuit in Fig. 11-12c shows a circuit for controlling the rotation. Adjusting the RATE potentiometer sets the rate of rotation. The logic level presented to the DIR (direction) input determines whether the figure rotates in a clockwise or counter clockwise direction.

So how can this 90°-shift procedure be used for simplifying the rotation of figures through much smaller increments? Rotating a figure through 45° increments is first a matter of working out two D-input programs, one for the basic 0° position and another for the 45° position. One additional rotation bit is required for selecting one of the two basic figures, while the two shown in Fig. 11-12 are necessary for determining how much either figure is shifted.

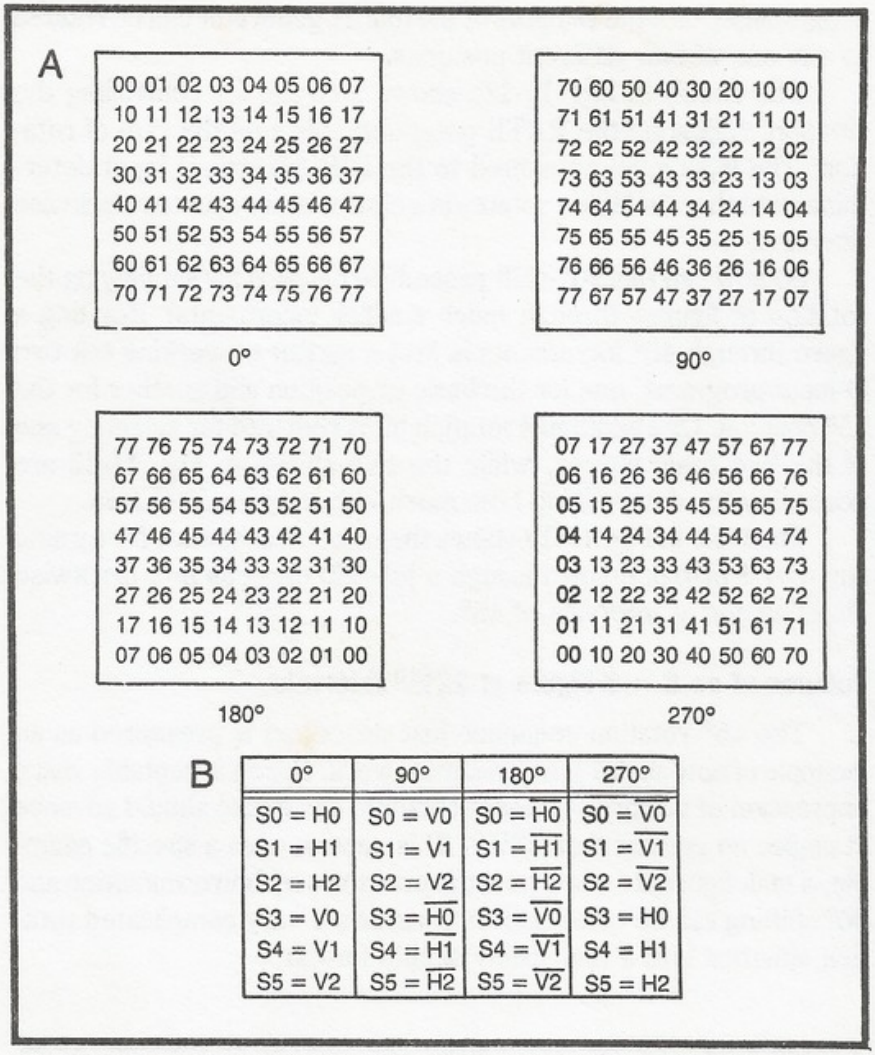
The table in Fig. 11-13 shows the rotation sequence for turning any 4 × 4 matrix figure through a full 360 degrees in a clockwise direction and at intervals of 45°.

Rotation of an 8 × 8 Figure at 22½° Intervals

The 45° rotation sequence just described is presented as an example of how such a scheme should work. For an acceptable visual impression of relatively smooth rotation, the figure should advance at angles no greater than 22½°. This section uses a specific example, a tank figure, to show how the principles of figure animation and 90° shifting can be combined to transform a very complicated rotation situation into a reasonably simple format.

ROTATION CONTROL			OPERATION	ROTATION EFFECT
B1	B0	B3		
0	0	0	0° IMAGE WITH 0° ROTATION	0°
0	0	1	45° IMAGE WITH 0° ROTATION	45°
0	1	0	0° IMAGE WITH 90° ROTATION	90°
0	1	1	45° IMAGE WITH 90° ROTATION	135°
1	0	0	0° IMAGE WITH 180° ROTATION	180°
1	0	1	45° IMAGE WITH 180° ROTATION	225°
1	1	0	0° IMAGE WITH 270° ROTATION	270°
1	1	1	45° IMAGE WITH 270° ROTATION	315°

Fig. 11-13. Table of rotation/frame selection for rotation a complex figure through 360 degrees at 45-degree increments.



0°	90°	180°	270°
S0 = H0	S0 = V0	S0 = $\overline{H0}$	S0 = $\overline{V0}$
S1 = H1	S1 = V1	S1 = $\overline{H1}$	S1 = $\overline{V1}$
S2 = H2	S2 = V2	S2 = $\overline{H2}$	S2 = $\overline{V2}$
S3 = V0	S3 = $\overline{H0}$	S3 = $\overline{V0}$	S3 = H0
S4 = V1	S4 = $\overline{H1}$	S4 = $\overline{V1}$	S4 = H1
S5 = V2	S5 = $\overline{H2}$	S5 = $\overline{V2}$	S5 = H2

Fig. 11-14. Matrix rationale for shifting 8×8 complex figures at 90-degrees. (a) 8×8 matrices with each cell showing the octal equivalent of the applied counting sequence. (b) Truth-table summary of addressing required for each 90-degree shift.

Figure 11-14 shows the 90° shifting rationale for an 8 × 8 matrix. The procedure for generating these matrices is identical to that of the simpler scheme in Fig. 11-11. The number of cells, however, has been increased substantially.

Using an 8 × 8 matrix generator calls for six select lines, designated S0 through S5. (Compare Fig. 11-14b and the matrix generator in Fig. 4-17.) Three of the select lines have inverted or noninverted H-counts applied to them, while the other three have inverted or noninverted V-counts connected to them. The 90° shifting is accomplished by varying this pattern of H- and V-count inputs as shown in Fig. 11-14b.

The circuit in Fig. 11-15, derived from the rationale in Fig. 11-14, can be considered a universal 90° rotator for 8 × 8 matrix figures. The circuit calls for six 4:1 multiplexers, each feeding the appropriate data to a select input on the matrix generator. The circuit satisfies the truth table in Fig. 11-15, and that table is based on the data derived from the basic 8 × 8, 90° shifting requirements in Fig. 11-14.

So much for the basic 8 × 8 90° shifter. Now consider how it can be combined with an animation sequence to rotate the figure of a tank on the screen.

Figure 11-16 shows a tank figure built within an 8 × 8 extended matrix. This is a 4-frame animation sequence that can be programmed at the D inputs of the matrix generator in Fig. 4-17.

Frame 1 shows the tank in its basic 0° position. This is probably the best tank figure that can be constructed within an 8 × 8 matrix. Frame 2 then shows the tank figure rotated clockwise by 22.5°. The image is terribly distorted, but it is the best we can do with a 64-cell matrix. And besides, the distortion really doesn't seem so bad to players lost in the action of a video game.

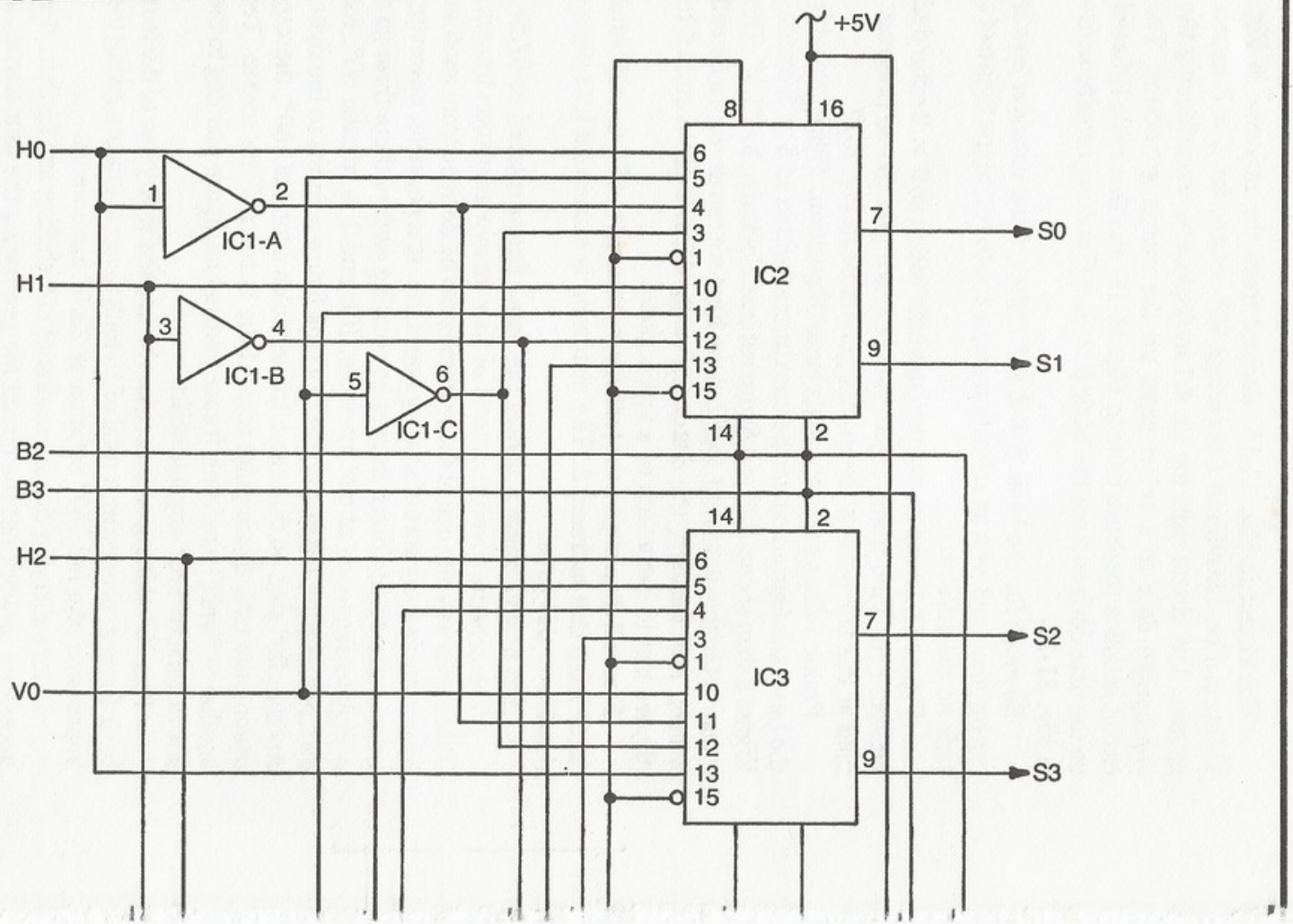
Frame 3 then shows the tank figure rotated to 45°, an additional 22.5° from that in frame 2. The distortion is somewhat less objectionable in this case.

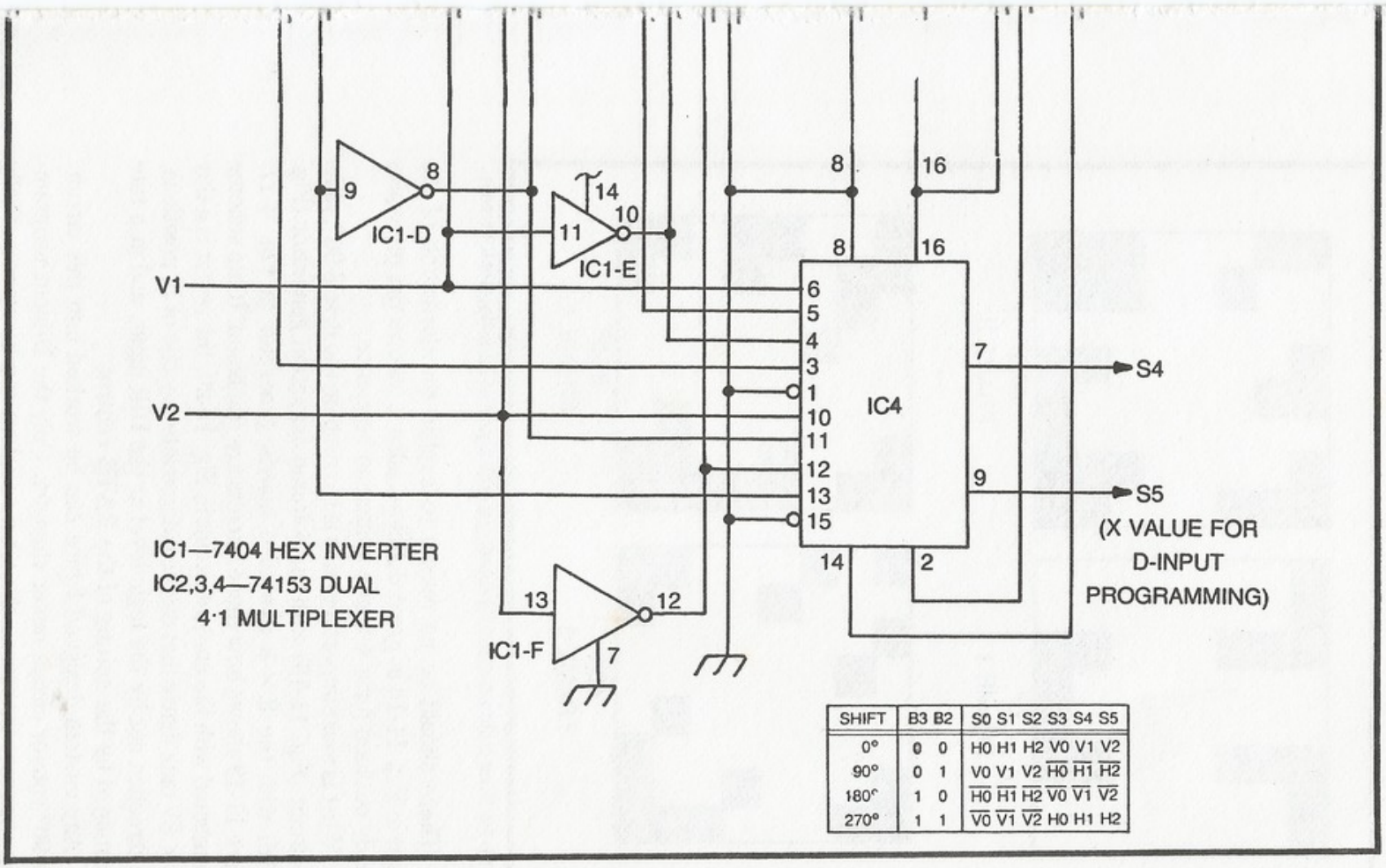
And finally, frame 4 shows the same figure rotated to 67.5°.

The scheme involves a total of four unique animation frames. There is no way any one of these frames can be derived from another by performing any sort of shifting operation. It is possible, however, to generate 12 additional frames by rotating each of them through a complete sequence of 90° intervals. If frame 1 is rotated 90°, for instance, the impression is that of a tank figure pointing to the right. Another 90° shift on that same frame yields a total of 180°, thereby making the tank figure point toward the bottom of the screen. Yet another 90° shift transforms frame 1 into a tank figure pointing to the left (a total of 270° angular shift).

This 90°-shifting sequence can be applied to any one of the four basic frames, producing a total of 16 tank images, images giving the impression of a full 360° rotation at 22.5° increments.

Before carrying the discussion of 90° shifting any further, it is necessary to work out the circuit for generating the four frames in Fig. 11-16. This is done using the same procedures outlined in the first part of this chapter. The data resulting from this procedure is summarized in the tables in Fig. 11-17. Figure 11-18 then shows the appropriate circuitry.


+5V
H0
H1
B2
B3
H2
V0
IC1-A
IC1-B
IC1-C
IC2
IC3
S0
S1
S2
S3



SHIFT	B3 B2	S0 S1 S2 S3 S4 S5
0°	0 0	H0 H1 H2 V0 V1 V2
90°	0 1	V0 V1 V2 $\overline{H0}$ $\overline{H1}$ $\overline{H2}$
180°	1 0	$\overline{H0}$ $\overline{H1}$ $\overline{H2}$ V0 V1 V2
270°	1 1	$\overline{V0}$ $\overline{V1}$ $\overline{V2}$ H0 H1 H2

Fig. 11-15 A universal 90-degree shifter for 8×8 matrix generators

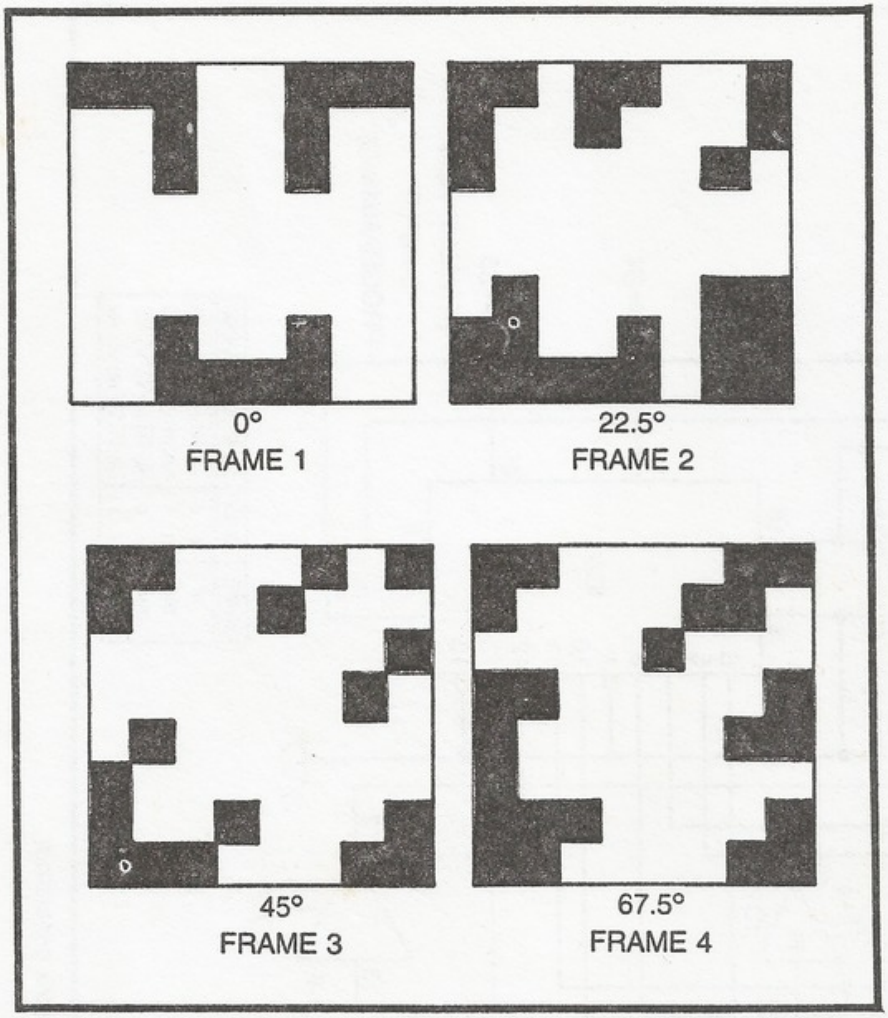


Fig. 11-16. Four critical frames for rotating a tank figure at 22.5-degree intervals.

There should be no reason to explain the derivation of the circuit in Fig. 11-18 in great detail because it follows the principles already outlined for a 4-frame animation sequence.

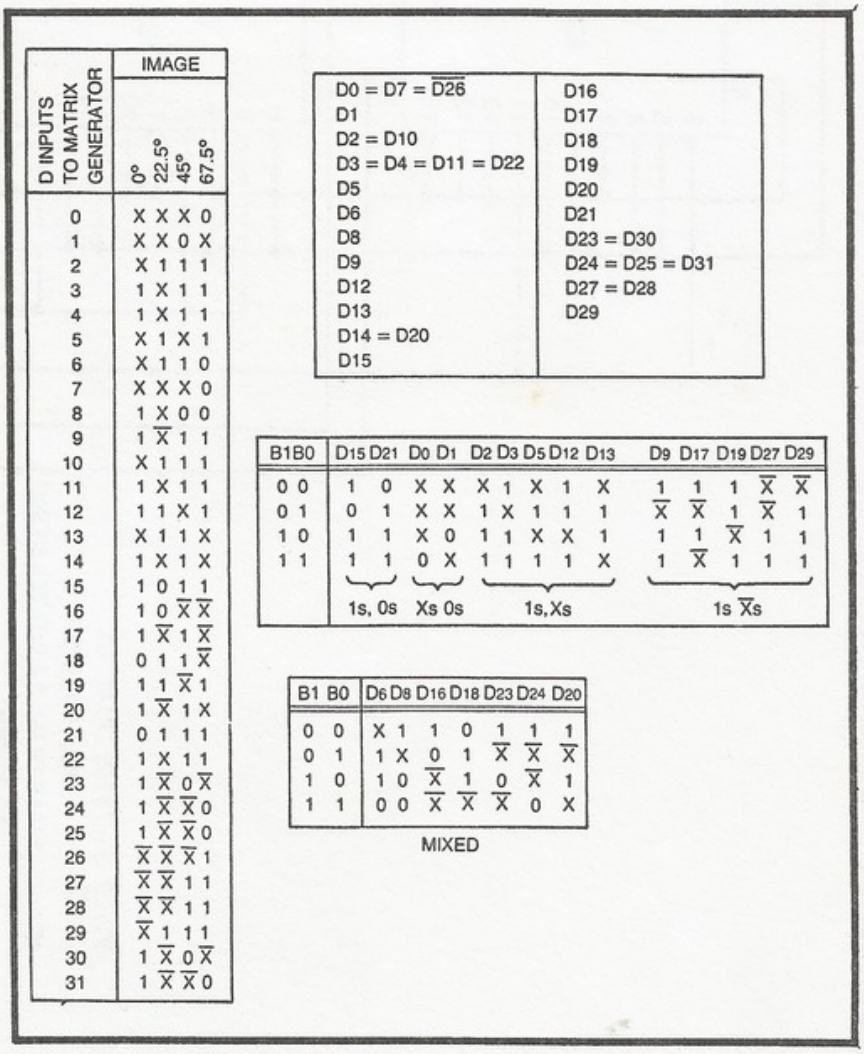
What is most important here is to combine the 8 × 8 90°, shifting circuit (Fig. 11-15) with the 4-frame animation generator (Fig. 11-18) with the 8 × 8 extended matrix generator in Fig. 4-17. Figure 11-19 shows how these circuits are combined. If the scheme is combined with the control circuit in Fig. 11-20, the result is a tiny 8H × 8V tank figure that can be rotated into any one of 16 positions, in a direction set by the logic level at the DIR input, and at a rate determined by the setting of the RATE control.

Any custom-designed figure can be worked into this circuit. The 90° rotator circuit never changes. Only the D-input programming changes. Of course the whole system can be systematically

expanded to accommodate more complex and less distorted figures. If it appears to an advanced experimenter that the complexity of the whole scheme is getting out of hand, he can resort to using a programmable read-only memory (PROM). This device is most often used by engineers who design high-quality commercial video games calling for fine rotation of complex figures.

COMBINING ROTATION AND FIGURE MOTION ACROSS THE SCREEN

All of the animation and rotation effects described thus far assume the figure will not move across the screen as the rotation takes place. It is often desirable to combine rotation and slipping-counter motion effects to heighten the impression of reality.



D INPUTS TO MATRIX GENERATOR	IMAGE 0°	22.5°	45°	67.5°
0	X	X	X	0
1	X	X	0	X
2	X	1	1	1
3	1	X	1	1
4	1	X	1	1
5	X	1	X	1
6	X	1	1	0
7	X	X	X	0
8	1	X	0	0
9	1	$\overline{X}$	1	1
10	X	1	1	1
11	1	X	1	1
12	1	1	X	1
13	X	1	1	X
14	1	X	1	X
15	1	0	1	1
16	1	0	$\overline{X}$	$\overline{X}$
17	1	$\overline{X}$	1	$\overline{X}$
18	0	1	1	$\overline{X}$
19	1	1	$\overline{X}$	1
20	1	$\overline{X}$	1	X
21	0	1	1	1
22	1	X	1	1
23	1	$\overline{X}$	0	$\overline{X}$
24	1	$\overline{X}$	$\overline{X}$	0
25	1	$\overline{X}$	$\overline{X}$	0
26	$\overline{X}$	$\overline{X}$	$\overline{X}$	1
27	$\overline{X}$	$\overline{X}$	1	1
28	$\overline{X}$	$\overline{X}$	1	1
29	$\overline{X}$	1	1	1
30	1	$\overline{X}$	0	$\overline{X}$
31	1	$\overline{X}$	$\overline{X}$	0

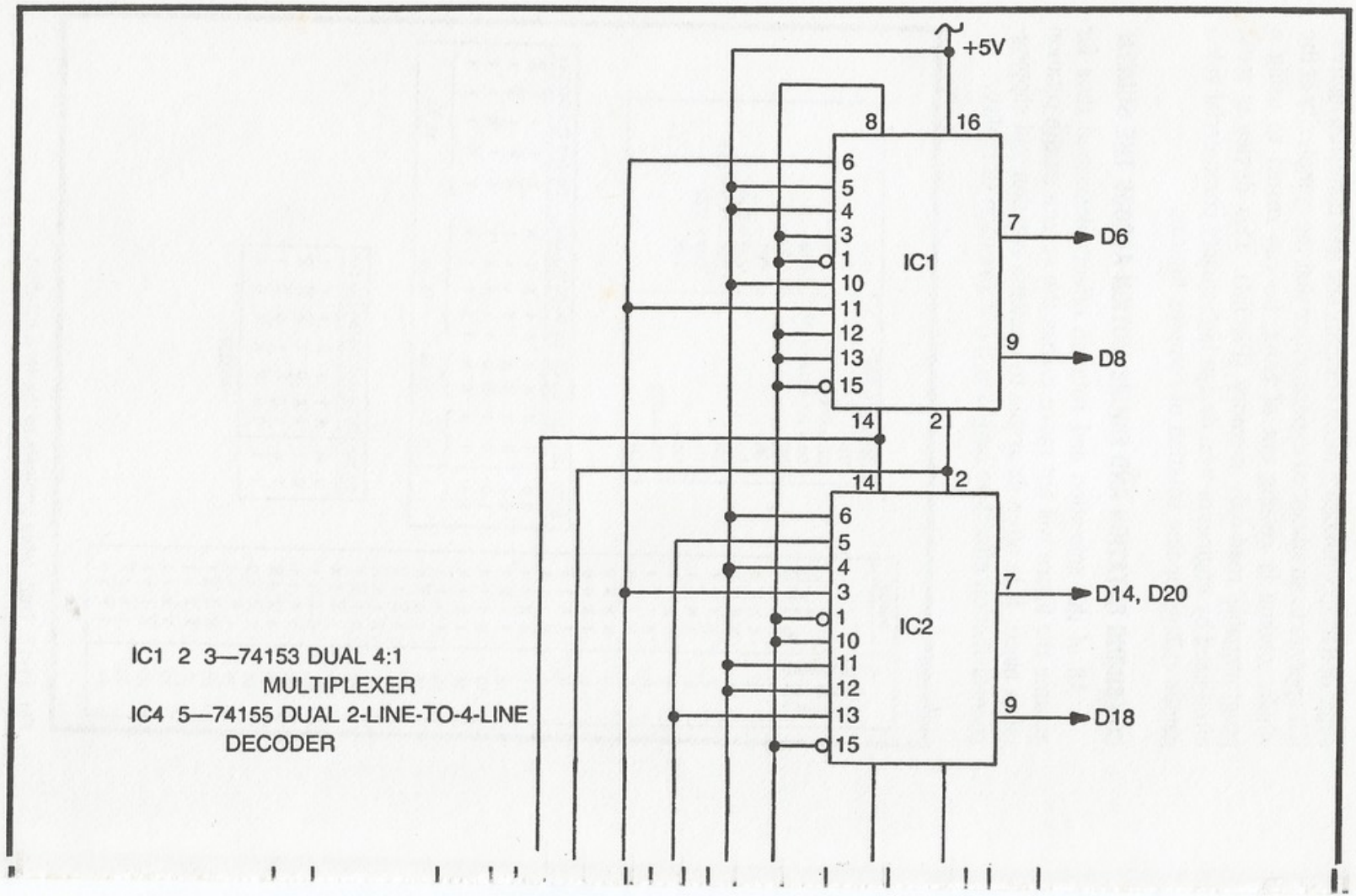
D0 = D7 = $\overline{D26}$	D16
D1	D17
D2 = D10	D18
D3 = D4 = D11 = D22	D19
D5	D20
D6	D21
D8	D23 = D30
D9	D24 = D25 = D31
D12	D27 = D28
D13	D29
D14 = D20	
D15	

B1B0	D15	D21	D0	D1	D2	D3	D5	D12	D13	D9	D17	D19	D27	D29
0 0	1	0	X	X	X	1	X	1	X	1	1	1	$\overline{X}$	$\overline{X}$
0 1	0	1	X	X	1	X	1	1	1	$\overline{X}$	$\overline{X}$	1	$\overline{X}$	1
1 0	1	1	X	0	1	1	X	X	1	1	1	$\overline{X}$	1	1
1 1	1	1	0	X	1	1	1	1	X	1	$\overline{X}$	1	1	1
	1s, 0s		Xs 0s		1s, Xs					1s $\overline{X}$s				

B1	B0	D6	D8	D16	D18	D23	D24	D20
0	0	X	1	1	0	1	1	1
0	1	1	X	0	1	$\overline{X}$	$\overline{X}$	$\overline{X}$
1	0	1	0	$\overline{X}$	1	0	$\overline{X}$	1
1	1	0	0	$\overline{X}$	$\overline{X}$	$\overline{X}$	0	X

MIXED

Fig. 11-17. Truth-table analysis of the tank rotation.



IC1 2 3—74153 DUAL 4:1
MULTIPLEXER
IC4 5—74155 DUAL 2-LINE-TO-4-LINE
DECODER

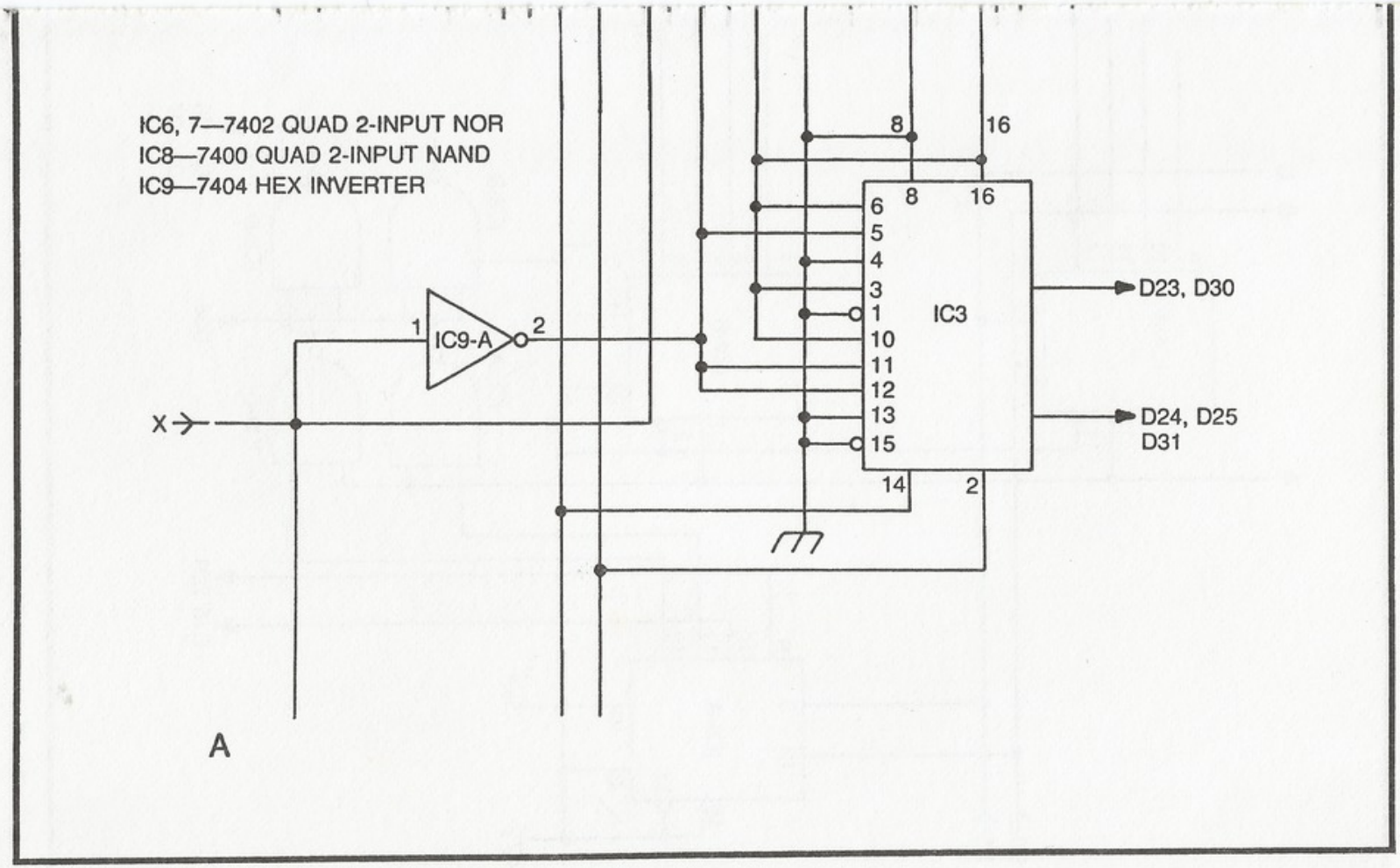


Fig. 11-18. Circuit for generating the D-input programming for Fig. 11-16

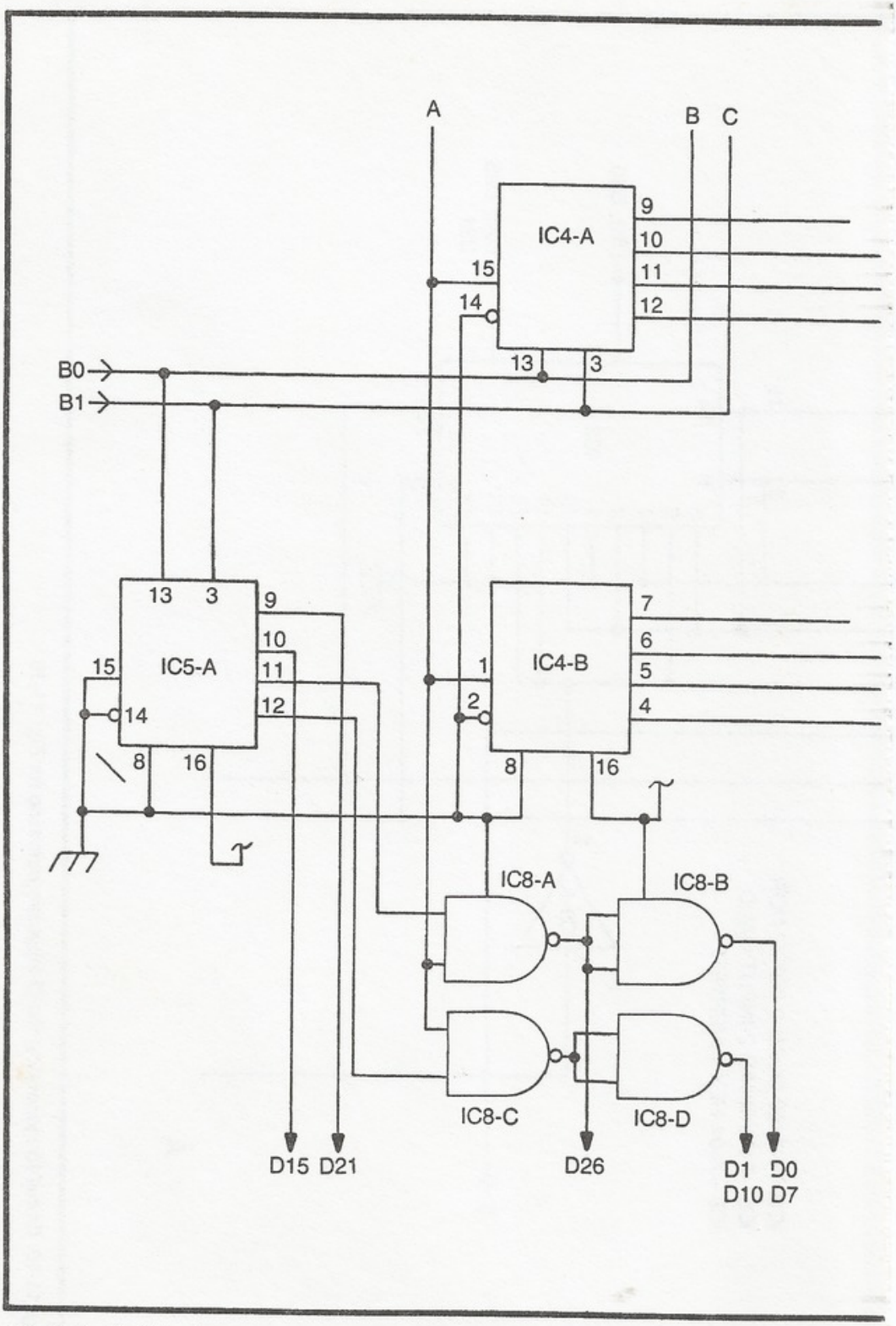


Fig. 11-8. Continued.

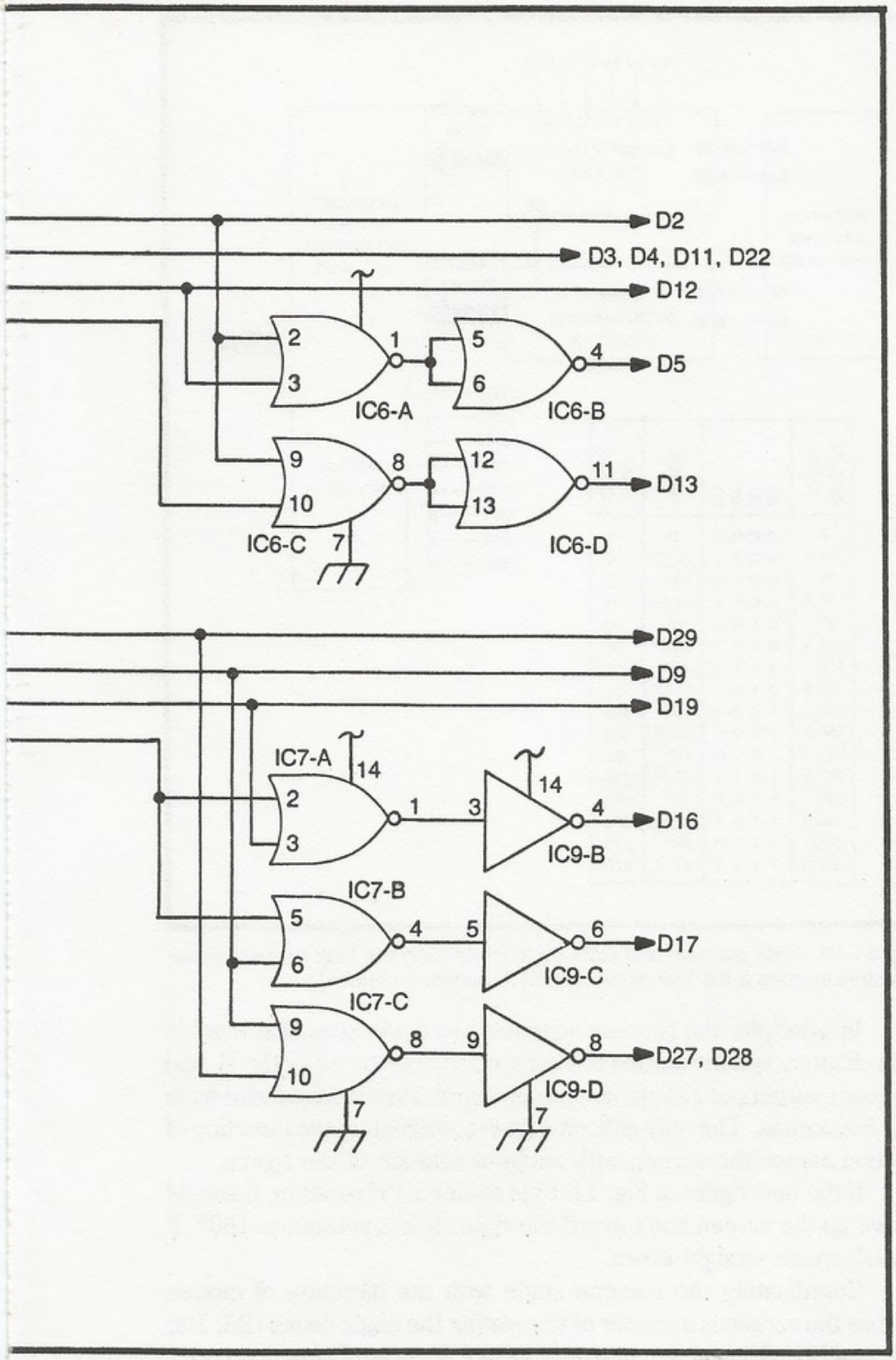
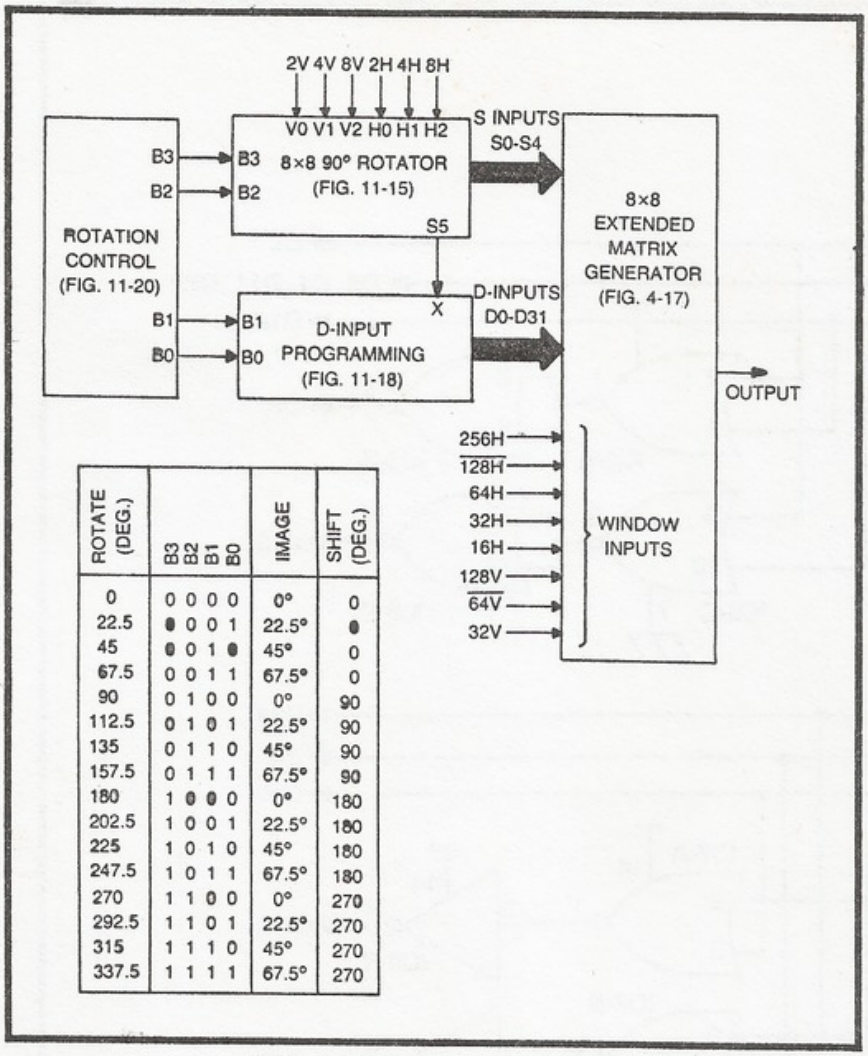


Fig. 11-8. Continued.



ROTATE (DEG.)	B3 B2 B1 B0	IMAGE	SHIFT (DEG.)
0	0 0 0 0	0°	0
22.5	● 0 0 1	22.5°	●
45	● 0 1 ●	45°	0
67.5	0 0 1 1	67.5°	0
90	0 1 0 0	0°	90
112.5	0 1 0 1	22.5°	90
135	0 1 1 0	45°	90
157.5	0 1 1 1	67.5°	90
180	1 ● ● 0	0°	180
202.5	1 0 0 1	22.5°	180
225	1 0 1 0	45°	180
247.5	1 0 1 1	67.5°	180
270	1 1 0 0	0°	270
292.5	1 1 0 1	22.5°	270
315	1 1 1 0	45°	270
337.5	1 1 1 1	67.5°	270

Fig. 11-19. Block diagram and code table for rotating the tank figure in either direction through a full 360 degrees at 22.5-degree increments.

In principle, the rotation schemes are devised as described in this chapter, and the motion effects are achieved by using the H- and V-count outputs of a slipping-counter board, rather than those from the Sourcebox. The only difficult part is coordinating the direction of motion across the screen with angle of rotation of the figure.

If the tank figure in Fig. 11-16 is set for a 45° rotation, it should move up the screen and toward the right. If it is rotated to 180°, it should move straight down.

Coordinating the rotation angle with the direction of motion across the screen is a matter of translating the angle codes (B3, B2, B1, and B0 in Fig. 11-18) into appropriate speed and direction codes for a set of horizontal and vertical slipping counters.

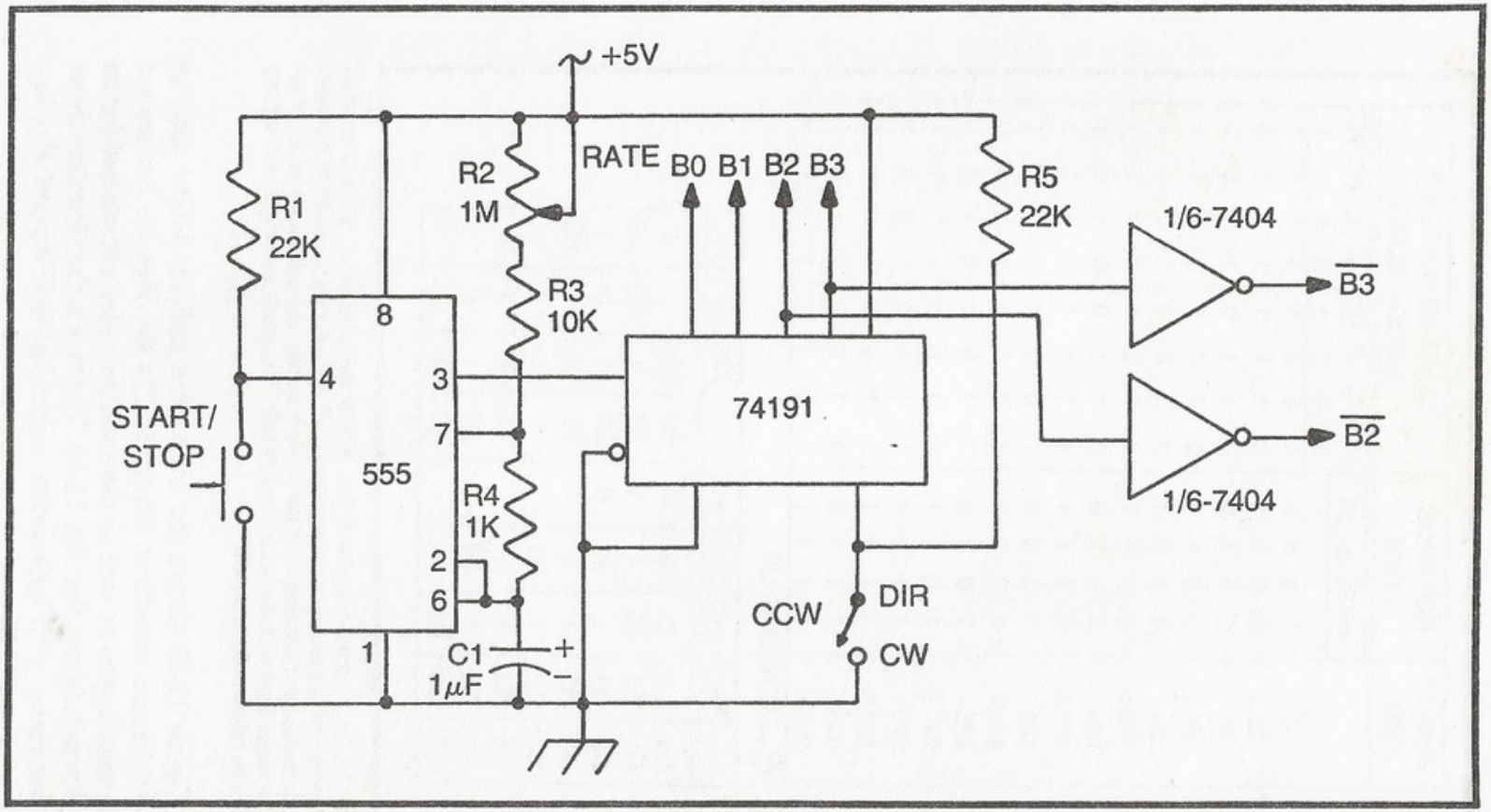


Fig. 11-20. Circuit for controlling the tank or similar 8×8 complex figure.

A

ANGLE* (DEG.)	ROTATION CODE				SLIPPING-COUNTER MOTION CODES							
	B3	B2	B1	B0	HC 8	HC 4	HC 2	HC 1	VC 8	VC 4	VC 2	VC 1
0	0	0	0	0	1	0	0	1	1	1	1	1
22.5	0	0	0	1	0	1	1	1	1	1	1	0
45	0	0	1	0	0	1	1	0	1	1	0	1
67.5	0	0	1	1	0	1	0	1	1	0	1	1
90	0	1	0	0	0	1	0	0	1	0	0	1
112.5	0	1	0	1	0	1	0	1	0	1	1	1
135	0	1	1	0	0	1	1	0	0	1	0	1
157.5	0	1	1	1	0	1	1	1	0	1	0	0
180	1	0	0	0	1	0	0	1	0	0	1	1
202.5	1	0	0	1	1	0	1	1	0	1	0	0
225	1	0	1	0	1	1	0	0	0	1	0	1
247.5	1	0	1	1	1	1	0	1	0	1	1	1
270	1	1	0	0	1	1	1	0	1	0	0	1
292.5	1	1	0	1	1	1	0	1	1	0	1	1
315	1	1	1	0	1	1	0	0	1	1	0	1
337.5	1	1	1	1	1	0	1	1	1	1	1	0

B *0° = STRAIGHT UP

	8HC	4HC	2HC	1HC	8VC	4VC	2VC	1VC
D0	1	0	0	$\overline{B2}$	$\overline{B3}$	$\overline{B3}$	1	1
D1	B3	$\overline{B3}$	1	1	$\overline{B3}$	1	$\overline{B3}$	B2
D2	B3	1	$\overline{B3}$	0	$\overline{B3}$	1	0	1
D3	B3	1	0	1	$\overline{B3}$	B3	1	$\overline{B2}$
D4	B3	1	B3		1	0	0	-
D5	B3	1	0		B3	$\overline{B3}$	1	-
D6	B3	1	$\overline{B3}$		B3	1	0	-
D7	B3	$\overline{B3}$	1	-	B3	1	B3	-

Fig. 11-21. Truth-table analysis of the relationships between angle of rotation and direction of motion across the screen. (a) Table of code translations between rotation codes and slipping-counter motion codes. (b) Summary of slipping-counter speed control codes as a function of D-input programming for translating them from rotation-code inputs.

Figure 11-21 shows the truth-table analysis behind such an angle-to-motion translator. Figure 11-21a specifies the desired set of rotation angles as well as the rotation codes established for an 8 × 8 matrix figure. See Fig. 11-13. The task is then to relate these rotation codes with appropriate horizontal and vertical slipping-counter-control codes.

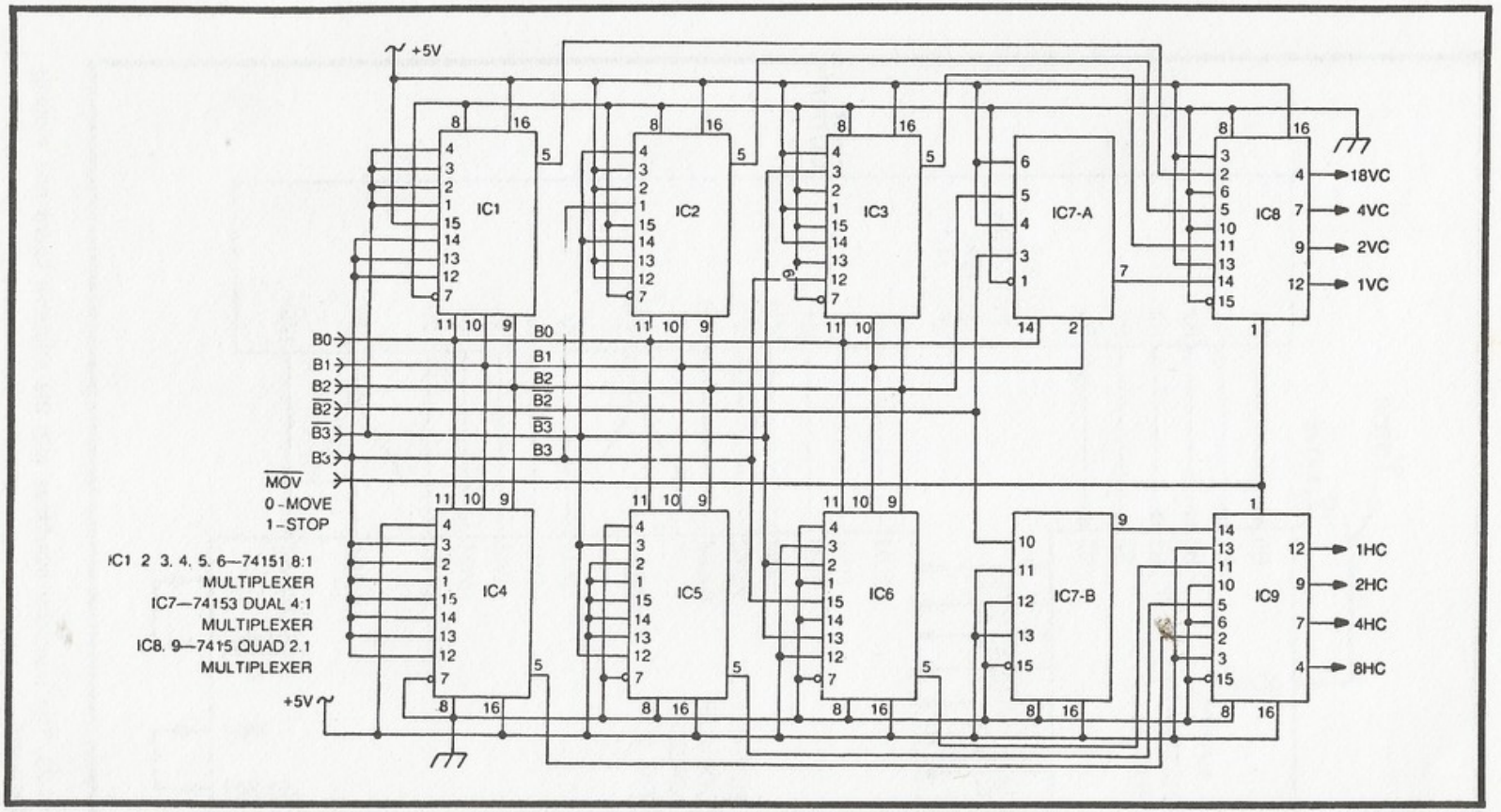


Fig. 11-22. Circuit for performing the translation between rotation and slipping-counter speed codes.

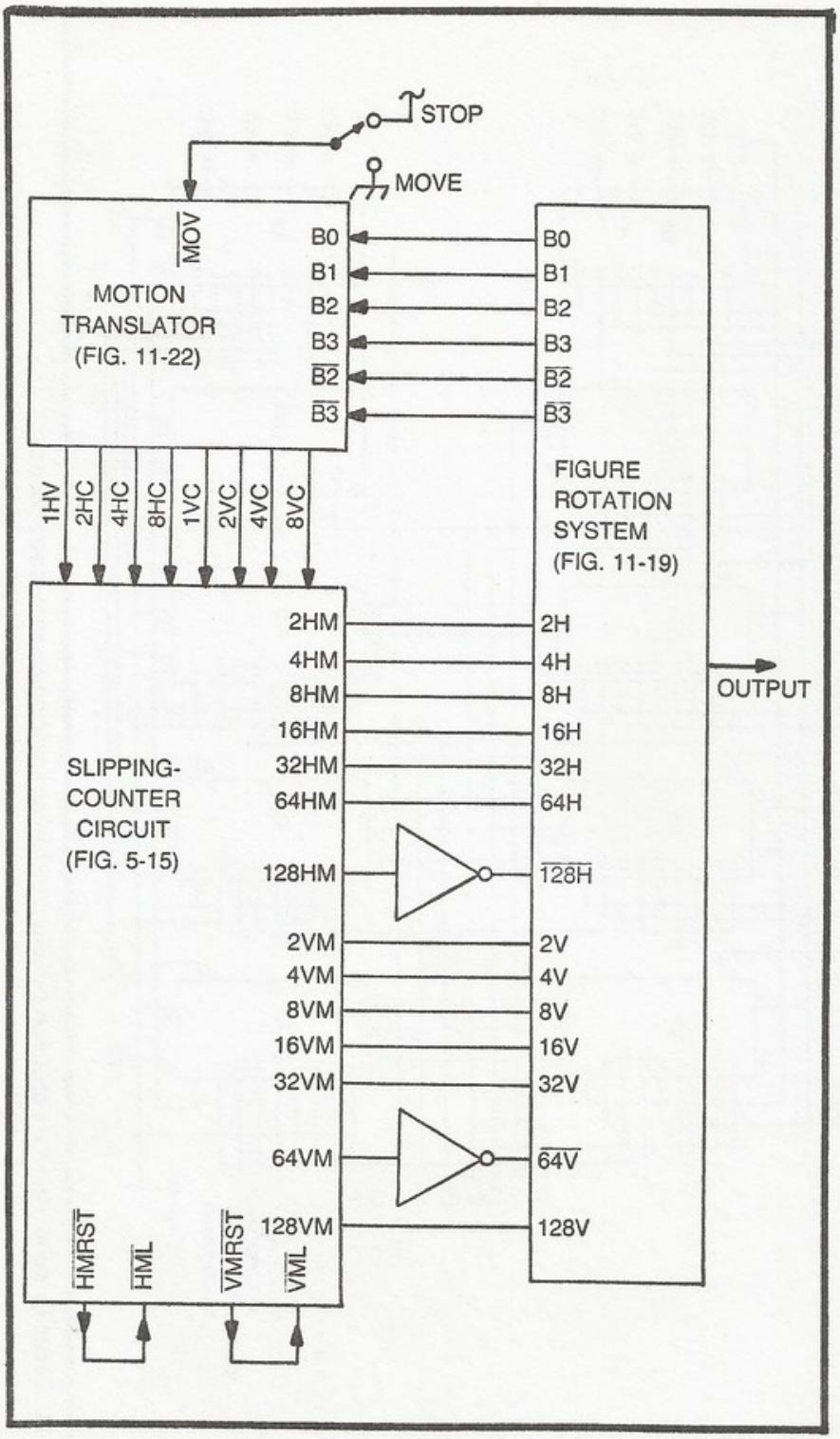


Fig. 11-23. The translator interfaces with the rotatable figure and slipping-counter board.

The general procedure is to determine the sine and cosine of the angles, then use the table in Fig. 7-22 to find the closest possible velocities for each. The sine of the angle determines the vertical velocity, while the cosine determines the horizontal velocity.

Table 11-21b shows the results of this translation process, assuming it will be carried out by means of a set of multiplexer circuits.

The resulting circuit diagram for the angle-to-motion translator is shown in Fig. 11-22. Figure 11-23 shows how the translator interfaces with the 8 × 8 rotatable figure and a slipping-counter board.

A similar approach using universal position programmers (Chapter 8) would provide a much wider range of figure speeds, but only at the cost of greater circuit complexity.