CHAPTER 11

DESIGN OF A SIMPLE SERIAL ARITHMETIC PROCESSOR

11.1 Introduction

For more complex digital circuits, the design techniques for combinational and sequential circuits are not applicable. The technique has to be elevated to another level. The basic elements are no longer just gates and flip-flops but include registers, counters, multiplexers, decoders, and other modular circuits. The function of a digital circuit is described by the transfer of information among storage elements or registers. Thus this level of design is known as the register transfer logic level.

In register transfer design, a digital system is usually partitioned into two distinct parts: data path and control circuit. A data path is the interconnections of basic elements. Data transfer or operations take place in the data path. Commands generated by a control circuit will instruct the data path what operations or data transfer to perform.

In this chapter, a simple serial arithmetic processor will be used as an example to illustrate the design at the register transfer level.

11.2 Adder

A half adder is a circuit that adds two single bits, d_1 and d_0, and generates a sum bit y_0 and a carry bit y_1 at the output. Table 11.1 and Figure 11.1 are the truth table and circuit for a half adder.

Table 11.1 Truth table for a half adder.

d_1	d_0	y_1	y_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

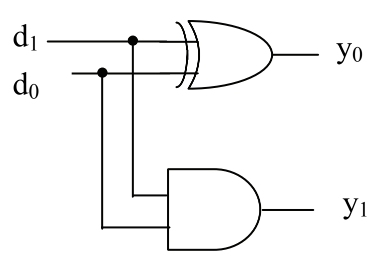


Figure 11.1 A half adder.

The process of adding two 4-bit numbers $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ is shown as follows. When adding b_0 to a_0, a sum bit S_0 and a carry bit c_1 are generated. The carry c_1

is then added to a_1 and b_1 to generate a sum bit S_1 and a carry bit c_2. The addition is complete after the same process is repeated two more times. The result is a 4-bit sum $S_3S_2S_1S_0$ and a carry c_4 from bit position 3.

$$\begin{array}{rcccc} & \boxed{c_3 & c_2 & c_1} & \\ & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline c_4 & S_3 & S_2 & S_1 & S_0 \end{array} \longleftarrow \text{carries generated from addition}$$

If an initial carry c_0 is included in the addition of A and B, the result remains unchanged if c_0 is equal to 0. The process now becomes identical for all the four bit positions, that is

$$\begin{array}{rr} & c_i \\ & a_i \\ + & b_i \\ \hline c_{i+1} & S_i \end{array}$$

for i = 0, 1, 2, 3. A combinational circuit for the addition of three single bits is known as a full adder. The truth table in Table 1.2 actually is the truth table for the full adder. y_1 is the carry bit c_{i+1} and y_0 is the sum bit S_i. A Boolean expression for S_i has been derived in Example 5.6.

$$S_i = a_i \oplus b_i \oplus c_i$$

The canonical sum-of-products expression for the carry bit c_{i+1} is

$$c_{i+1} = a_i' \, b_i \, c_i + a_i \, b_i' \, c_i + a_i \, b_i \, c_i' + a_i \, b_i \, c_i$$

The simplest sum-of-products expression is

$$c_{i+1} = a_i \, b_i + b_i \, c_i + a_i \, c_i \tag{11.1}$$

A different expression for the carry bit can be obtained as follows:

$$\begin{aligned} c_{i+1} &= a_i' \, b_i \, c_i + a_i \, b_i' \, c_i + a_i \, b_i \, c_i' + a_i \, b_i \, c_i \\ &= (a_i' \, b_i + a_i \, b_i')c_i + a_i \, b_i \, (c_i' + c_i) \\ &= (a_i \oplus b_i) \, c_i + a_i \, b_i \qquad (11.2) \end{aligned}$$

Implementation of c_{i+1} using the simplest sum-of-products expression in Equation (11.1) requires three 2-input AND gates and one 3-input OR gate. However, the expression in

Equation (11.2) needs only two 2-input AND gates and one 2-input OR gate. ($a_i \oplus b_i$) is available from the implementation of S_i. The circuit for the full-adder is shown in Figure 11.2, which can be considered as two half adders. The two half adders are indicated by different gray levels. The carry output of the full adder is obtained by ORing the carry outputs of the two half adders.

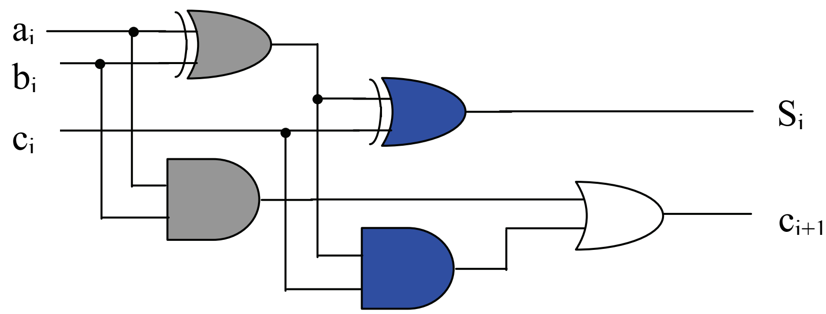


Figure 11.2 A full adder.

The addition of two n-bit numbers and an initial carry c_0 is given below.

$$
\begin{array}{cccccccc}
 & & & & & & & c_0 \\
 & a_{n-1} & a_{n-2} & \ldots\ldots\ldots\ldots & a_2 & a_1 & a_0 & \\
+ & b_{n-1} & b_{n-2} & \ldots\ldots\ldots\ldots & b_2 & b_1 & b_0 & \\
\hline
c_n & S_{n-1} & S_{n-2} & \ldots\ldots\ldots\ldots & S_2 & S_1 & S_0 &
\end{array}
$$

Addition can be carried out by a circuit shown in Figure 11.3. The circuit, known as a ripple-carry adder, consists of n full adders. The addition in each bit position is performed by one full adder. Assume that the propagation delay in each full adder is τ. The total propagation delay in the ripple-carry adder is $n\tau$. The design of a ripple-carry adder is simple. But it takes longer to get the sum because of the propagation delay of the carry from the least significant bit to the most significant bit. Because adder is an important circuit in digital systems, there are many other designs that try to improve its efficiency by reducing the propagation delay. Ripple-carry adder is a parallel adder. Since all the n full adders in a ripple-carry adder are identical, addition can be carried out by using just one full adder and by repeating the addition n times using the same full adder.

The circuit in Figure 11.4 shows the addition of two 4-bit numbers, A and B, using only one full adder. In the beginning, two 4-bit numbers $a_3a_2a_1a_0$ and $b_3b_2b_1b_0$ are stored in the two 4-bit universal shift registers R_1 and R_2. An initial carry c_0 is also stored in the D flip-flop. In each clock cycle during the addition, the contents of the registers are shifted to the right one bit. The sum bit generated by the full adder is shifted into the leftmost position of R_1. The rightmost bit of R_2 is rotated into the leftmost position. The carry generated by the full adder is stored in a D flip-flop. Q becomes the carry input to the full adder in the next clock cycle. It takes four clock cycles to complete the addition.

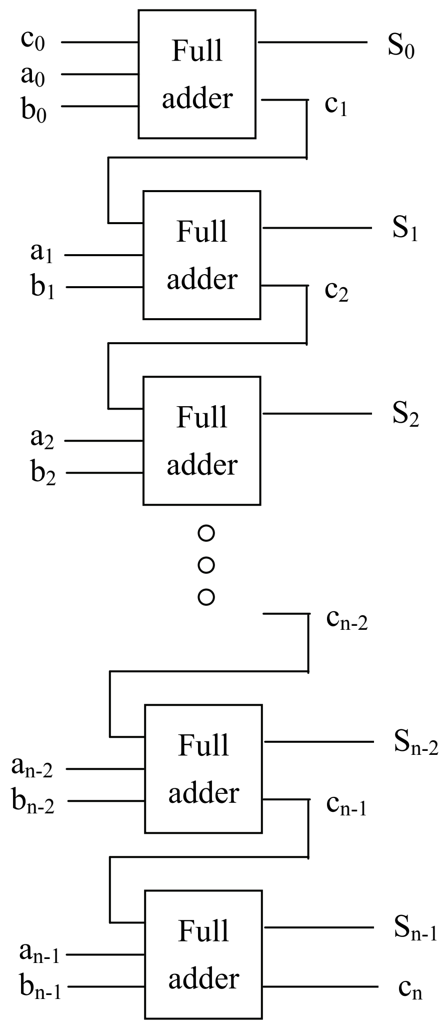


Figure 11.3 A ripple-carry adder.

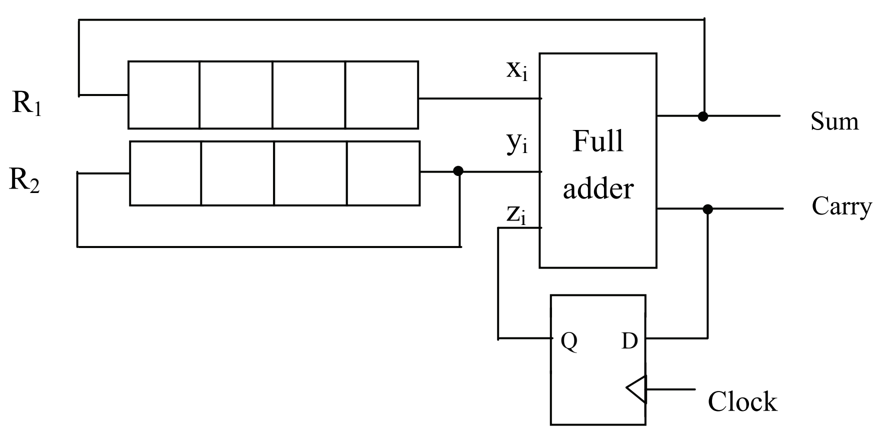


Figure 11.4 A serial adder.

At the completion of addition, the sum $S_3S_2S_1S_0$ is stored in R_1. c_4 is stored in the D flip-flop. The contents of R_2 remain unchanged. Table 11.2 lists the contents of R_1, R_2, Q and D of the D flip-flop, the inputs and outputs of the full adder for each clock cycle. Since addition is performed for one bit position in each clock cycle. The adder is called a serial adder.

Table 11.2 Contents of the serial adder.

Clock cycle	R_1	R_2	x_i	y_i	z_i (Q)	Sum	Carry (D)
0	$a_3\, a_2\, a_1\, a_0$	$b_3\, b_2\, b_1\, b_0$	a_0	b_0	c_0	S_0	c_1
1	$S_0\, a_3\, a_2\, a_1$	$b_0\, b_3\, b_2\, b_1$	a_1	b_1	c_1	S_1	c_2
2	$S_1\, S_0\, a_3\, a_2$	$b_1\, b_0\, b_3\, b_2$	a_2	b_2	c_2	S_2	c_3
3	$S_2\, S_1\, S_0\, a_3$	$b_2\, b_1\, b_0\, b_3$	a_3	b_3	c_3	S_3	c_4
4	$S_3\, S_2\, S_1\, S_0$	$b_3\, b_2\, b_1\, b_0$	S_0	b_0	c_4	N/A	N/A

11.3 Signed Numbers

The binary numbers introduced in Chapter 2 are unsigned numbers. All unsigned numbers are positive numbers. Signed numbers can be either positive or negative. Two signed number representations are introduced in this section.

Sign-magnitude representation is the simplest of all signed number representations. For an n-bit number, the leftmost bit is assigned as a sign bit. When the sign bit is 0, the number is positive. The sign bit of a negative number is 1. The other (n-1) bits are used for the magnitude of the number. Thus the range of an n-bit signed number N using sign-magnitude representation is

$$-(2^{n-1}-1) \leq N \leq (2^{n-1}-1)$$

Some examples of sign-magnitude representation are given below.

0 1 0 0 1 0 1 1	+ 75
0 1 1 1 1 1 1 1	+ 127
1 1 1 1 1 1 1 1	− 127
1 0 0 0 0 0 0 1	− 1
0 0 0 0 0 0 0 0	+ 0
1 0 0 0 0 0 0 0	− 0

It is seen that there are two different representation of 0: +0 and −0. Although sign-magnitude representation is simple in representing signed numbers, it is not very useful when it is applied to arithmetic operations in computers.

Two's (2's) complement representation for signed numbers in computers is popular. How signed numbers are represented in the 2's complement system is shown graphically in Figure 11.5 for a 4-bit signed number. Similar to sign-magnitude representation, the leftmost bit is again used as a sign bit. The magnitude of a positive number is the same as sign-magnitude representation. Negative numbers are represented by the combinations from 1000 to 1111. Their decimal equivalents, however, are not from −0 to −7, but from −8 to −1. The assignment of decimal equivalents is in a direction opposite to that of sign-magnitude representation. Also there is only one representation for 0, which frees up 1000 to be assigned to a negative number. Thus the range of an n-bit signed number N in the 2's complement system is

$$-2^{n-1} \leq N \leq (2^{n-1} - 1)$$

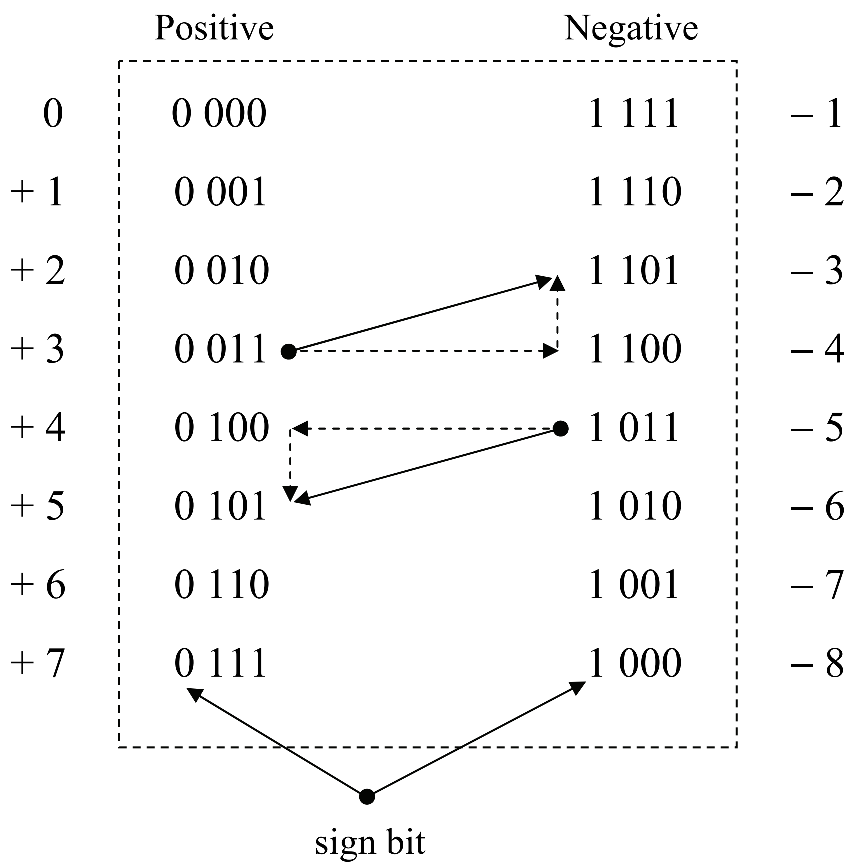


Figure 11.5 Two's complement representation for 4-bit signed numbers.

Negating a signed number in sign-magnitude representation can be easily performed by just complementing the sign bit. To negate a signed number in the 2's complement system, every bit in the number is inverted and one is added after the inversion. Negation is illustrated by two examples in Figure 11.5. + 3 is negated to –3. –5 is changed to +5. Bit inversion is indicated by a horizontal directed line. A vertical directed line indicates the addition of one.

The bit inversion of an n-bit number $Y = y_{n-1}\, y_{n-2}\ \ldots\ldots\ y_2 y_1 y_0$ is equivalent to the subtraction of an n-bit number $2^n - 1$ by Y, which is shown below.

$$
\begin{array}{rcccccc}
 & 1 & 1 & \ldots\ldots & 1 & 1 & 1 \\
- & y_{n-1} & y_{n-2} & \ldots\ldots & y_2 & y_1 & y_0 \\
\hline
 & y_{n-1}' & y_{n-2}' & \ldots\ldots & y_2' & y_1' & y_0'
\end{array}
$$

In each of the n bit positions, 1 is the minuend and y_i the subtrahend. Because $1 - 1 = 0$ and $1 - 0 = 1$, $1 - y_i = y_i'$. $y_{n-1}'\, y_{n-2}'\ \ldots\ldots\ y_2'\, y_1'\, y_0'$ is called the one's (1') complement of Y and can be written as ${}^1Y$. ${}^2Y$, the 2's complement of Y, is defined as

$${}^2Y = {}^1Y + 1$$

Thus the 2's complement of Y can also be obtained by subtracting Y from 2^n because

$${}^2Y = {}^1Y + 1 = (2^n - 1) - Y + 1 = 2^n - Y$$

In computer arithmetic, the subtraction of two n-bit signed numbers, $A - B$, can be carried out by adding the negative of the subtrahend to the minuend, i.e. $A - B = A + (-B)$. When $-B$ is replaced with ${}^2B$, the subtraction becomes

$$A + {}^2B = A + (2^n - B) = (A - B) + 2^n$$

The result seems to be incorrect because of the extra term 2^n. In fact, this extra term does not have any effect on the result of $(A - B)$. Numbers are stored in fixed-length registers or memory locations in computers. When a number exceeds the length of a register or memory location, the extra bits of the number are left out. Since the binary equivalent of 2^n is an (n+1)-bit number 100.....000, the leftmost bit which is 1 cannot be stored in an n-bit register. Thus 2^n is truncated to an n-bit number which is 00....000. Two examples are given below to show the subtraction of two 4-bit signed numbers in 2's complement arithmetic.

Decimal : $5 - 2 = 5 + (-2) = +3$

2's complement : $0101 - 0010 = 0101 + (-\,0010)$

$$\Downarrow \text{ conversion to 2's complement}$$

$$0101 + 1110 = 1\ 0011$$

$$\Downarrow \text{ discard of extra bit}$$

$$0011$$

The first conversion sign $\Downarrow$ is to replace – 0010 with its 2's complement. The second conversion sign is to show the discard of the extra bit, which is the carry from bit position 3.

Decimal : $2 - 5 = 2 + (-5) = -3$

2's complement : $0010 - 0101 = 0010 + (-0101)$

$\Downarrow$ conversion to 2's complement

$0010 + 1011 = 0\ 1101$

$\Downarrow$ discard of extra bit

1101

The result is a negative number. The 2's complement of this number is 0011.

In the addition or subtraction of two n-bit signed numbers, the correct result sometimes may require a size of more than n bits. Such a situation is known as "overflow" if the result is greater than $(2^{n-1} - 1)$ and "underflow" if it is smaller than -2^{n-1}. The result is meaningless when overflow or underflow occurs. An example of adding 5 to 4, or 0100 + 0101, is given as an illustration for overflow. All numbers have a length of four bits. The addition yields a sum of 1001 with a carry of 0 from bit position 3. The result from the addition of two positive numbers is positive. However, the sign bit of the sum 1001 indicates that it is negative. This is an overflow situation.

11.4 Algorithmic State Machine (ASM) Chart

Similar to the flow chart used to describe a software algorithm that can be programmed and implemented on a computer, algorithmic state machine (ASM) chart is a technique used to describe a sequence of actions carried out in a sequential system. There are three basic elements for ASM charts: state box, decision box, and conditional output box, which are shown in Figure 11.6.

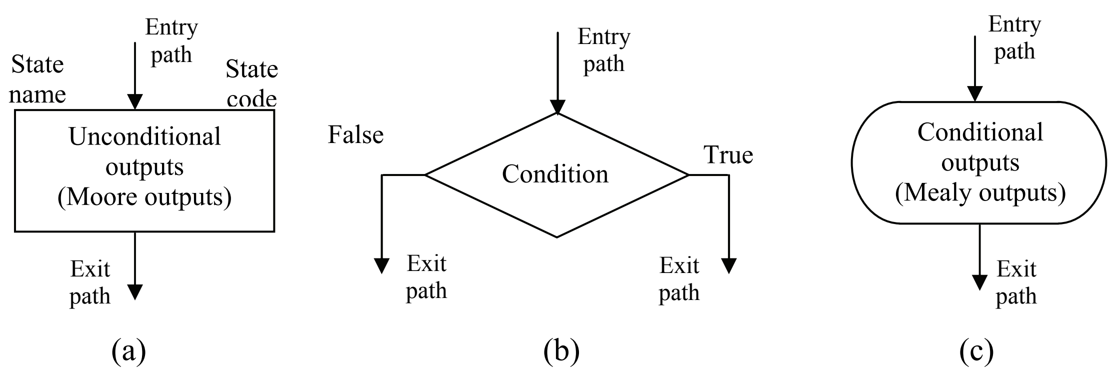
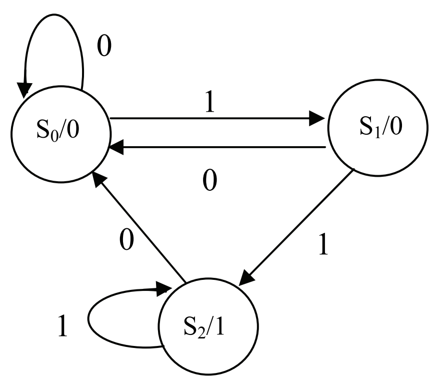


Figure 11.6 Basic elements of ASM charts. (a) State box. (b) Decision box. (c) Conditional output box.

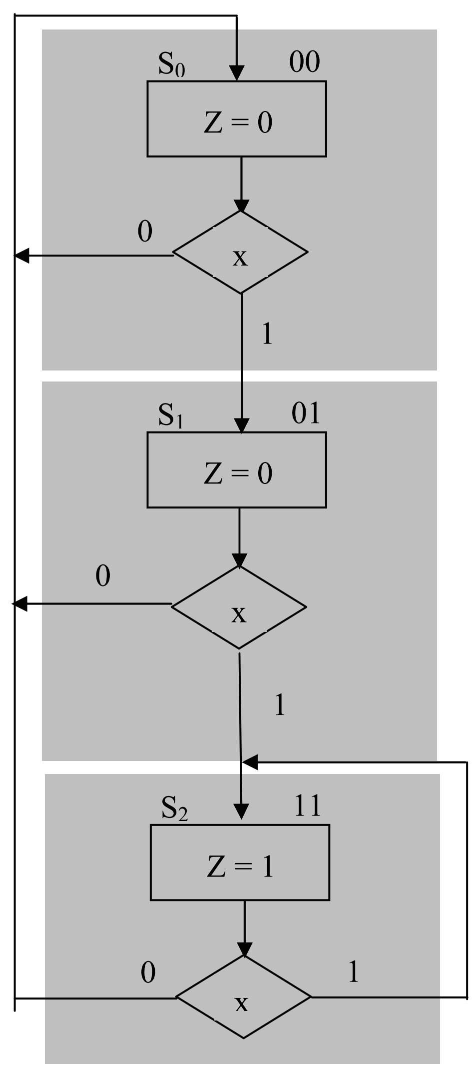
In an ASM chart, the basic elements can be structured into a number of blocks. The actions taken place in each state are described within a block and carried out in one clock cycle. There is always one and only one state box for each block. The symbol for a state box is a rectangle. The name and the code of a state are placed, respectively, at the upper left or right corner outside the box. Unconditional operations to be executed and unconditional outputs to be produced in a state are described inside the state box. Outputs produced in a Moore model state machine are unconditional. A decision box is for the testing of a certain condition or value. A diamond-shaped box is used. There are two exit paths, one for true and the other for false, or one for a value of 1 and the other for a value of 0. There may be more than one decision box or no decision box at all within a block. An oval-shaped box is used for a conditional output box. The actions described inside a conditional output box will be executed only under certain conditions. Outputs produced in a Mealy model state machine are conditional. The simplest block consists of just a state box. The translation of a Moore model state diagram to an ASM chart is shown in Figure 11.7. Each block is highlighted. The conversion of a Mealy model state diagram to an ASM chart is shown in Figure 11.8.



(a)

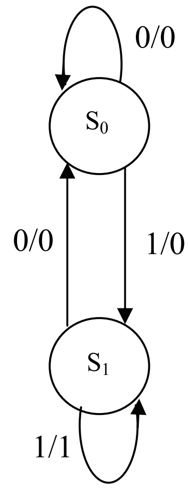
State name	State code Q_1Q_0
S_0	0 0
S_1	0 1
S_2	1 1

(b)



(c)

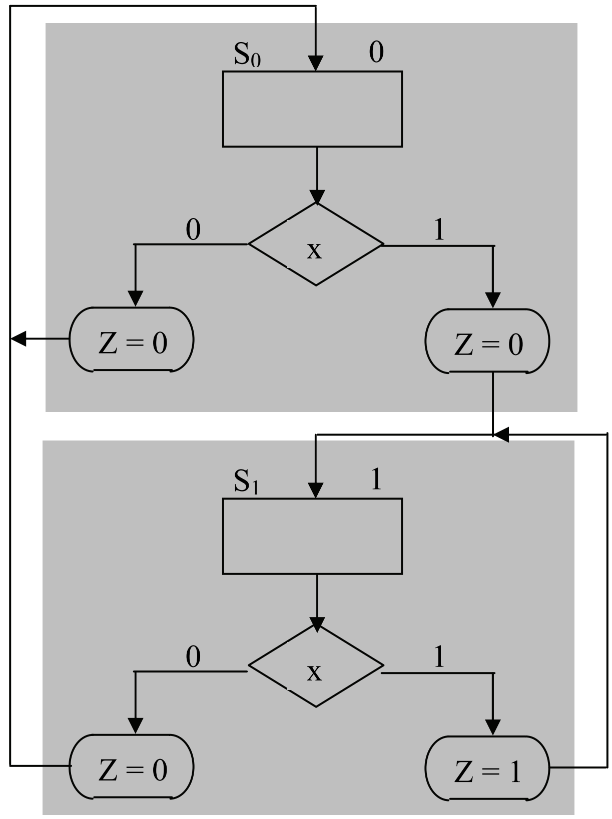
Figure 11.7 Conversion of a Moore state diagram to ASM chart. (a) State diagram. (b) State assignment. (c) ASM chart.



(a)

State name	State code Q
S_0	0
S_1	1

(b)



(c)

Figure 11.8 Conversion of a Mealy state diagram to ASM chart. (a) State diagram. (b) State assignment. (c) ASM chart.

11.5 A Simple Serial Arithmetic Processor

Data Path

A simple arithmetic processor to perform A + B and A − B serially for two 4-bit signed numbers is introduced in this section. The serial adder in Figure 11.4 is used in the design. The data path of the processor is given in Figure 11.9. The two 4-bit signed numbers A and B are stored in the shift registers R_1 and R_2 respectively. Loading of an external 4-bit data into a register is executed by asserting the control signal "Load1" or "Load2". In each clock cycle of an arithmetic operation, the contents in R_1 and R_2 are shifted right one bit. Shifting is controlled by a control signal "Shift". When "Hold1" or "Hold2" is asserted, the contents of R_1 or R_2 remain intact. Arithmetic operation is selected by an external input OPCODE. The operation is A + B when OPCODE = 0. A − B is executed when OPCODE = 1. For subtraction, the serial adder will perform the addition of B or ^{2}B to A. To obtain ^{2}B, b_i, the rightmost bit from R_2, should be complemented before it can be inputted to the adder at y and the carry c_0 is initialized to a value of 1. For A + B, $c_0 = 0$ and b_i is the input to the adder at y. The input circuit is used to select either b_i or b_i'. Note that the rightmost bit of R_2 is referred to as b_i because R_2 is shifted to the right to provide b_1, b_2, b_3 to the input circuit in the next three clock cycles. Thus

$$y = \text{OPCODE} \oplus b_i$$

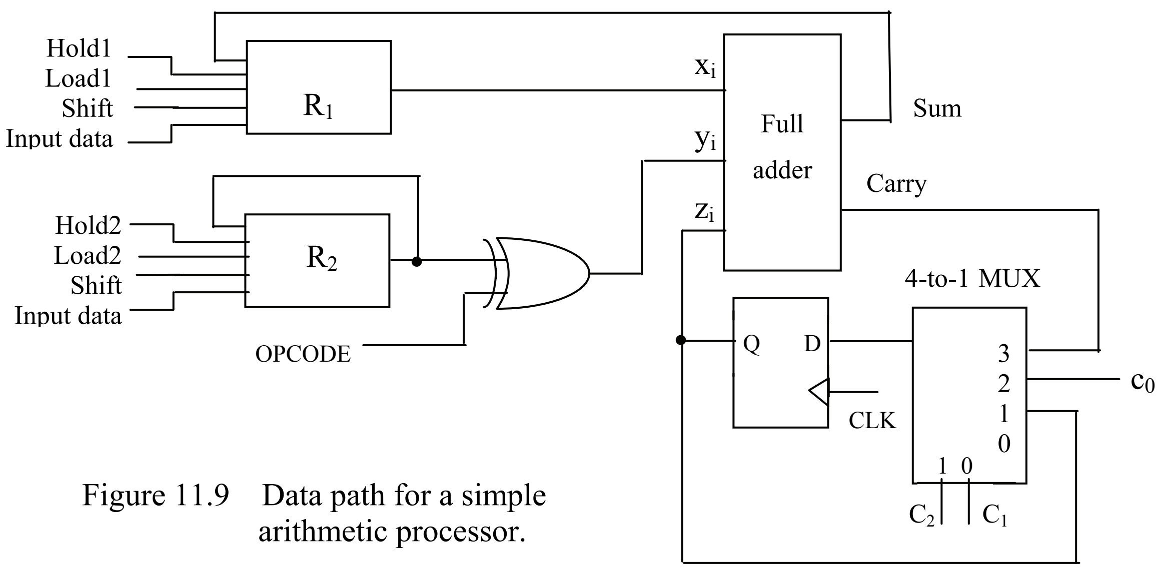


Figure 11.9 Data path for a simple arithmetic processor.

A 4-to-1 multiplexer is used for the selection of the initial carry c_0, the carry generated by the full adder, or Q. When no arithmetic operation is performed, the value stored in the D flip-flop remains unchanged even if the flip-flop is triggered by a clock pulse. In such a case, the value at Q is re-loaded.

ASM Chart

The operations of the arithmetic processor are described by an ASM chart in Figure 11.10. Before any operation takes place, the circuit is reset to the initial state T_0 by a RESET input pulse. When the START input is 0, the circuit remains idle and stays in T_0. The contents of R_1, R_2, and Q should remain unchanged. R_1 and Q may still have the results from the last arithmetic operation. Therefore Hold1 and Hold2 are asserted. In the ASM chart, the appearance of a signal name without a logic value suggested that the signal is asserted. Thus Shift, Load1, and Load 2 are not asserted in T_0 when START = 0. The value stored in the D-flip-flop is re-loaded with the selection of Q by the 4-to-1 multiplexer with $C_2C_1 = 01$.

Operations begin to take place when START = 1. A 4-bit signed number A is first loaded into R_1, which is followed by the loading of the second number B into R_2 in T_1. An initial carry is also stored in the D flip-flop in T_1. To store the two numbers in R_1 and R_2, the two control signals Load1 and Load2 should be asserted in T_0 and T_1 respectively. While loading the second number in T_1, Hold1 must be asserted. Otherwise the number just stored in R_1 may change. The contents of the two registers are shifted right 1-bit in each of the following four clock cycles. A control signal Shift is therefore generated by the control circuit to trigger the shifting of the contents in R_1 and R_2. The carry stored in the D flip-flop comes from the full adder output in these four clock cycles. Therefore control signals $C_2C_1 = 11$. The value of START is assumed to be don't-care after it becomes 1.

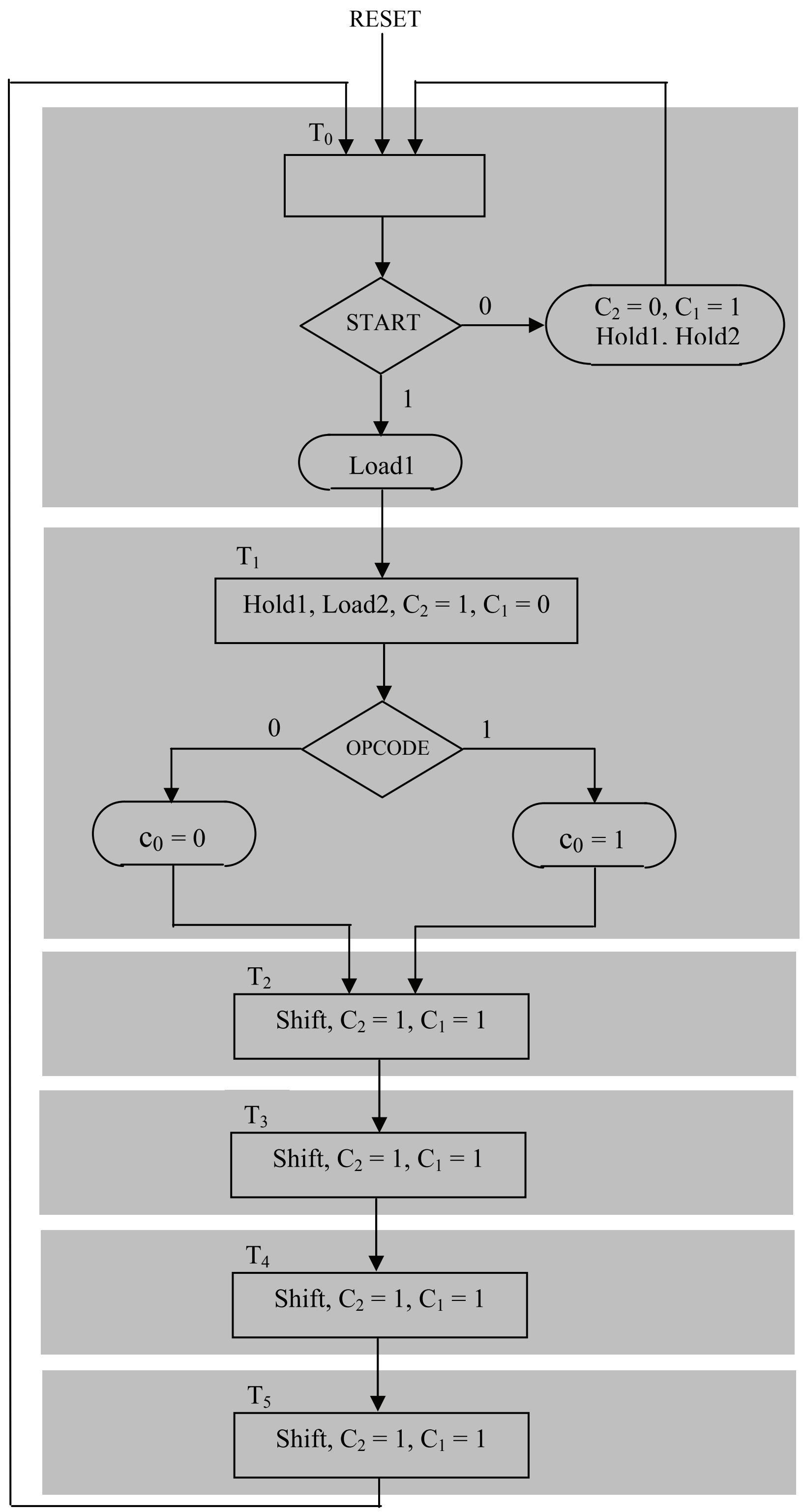


Figure 11.10 ASM chart for the arithmetic processor.

The operations in T_1 are used to explain the difference between an ASM chart and a flowchart. If Figure 11.10 were a flowchart, the values of Hold1, Load2, C_2, and C_1 are generated before that of c_0. In an ASM chart, they are generated at the same time by the control circuit. Since c_0 is the same as OPCODE that is an input to the arithmetic processor, c_0 may be generated before other control signals. Although the initial carry c_0 and the asserted Load2 are generated in T_1, loading c_0 and B into the D flip-flop and R_2 will not occur until T_2, that is, after the flip-flop and the register are triggered by a clock pulse.

Design of State Generator

The six states T_0, T_1, ..., T_5 are provided by a state generator. It is built on a 6-bit ring counter. The representation of each state by one flip-flop is also known as one-hot state assignment. The state assignment table is given in Table 11.3. A state table obtained from the ASM chart in Figure 11.10 is given in Table 11.4. When the state generator is in T_i, only $Q_i = 1$. Therefore T_i and Q_i will be used interchangeably. From the ASM chart, it is seen that the state generator does not advance automatically from one state to the next. The state generator remains in T_0 if START = 0.

Table 11.3 State assignment.

State	Q_0 Q_1 Q_2 Q_3 Q_4 Q_5
T_0	1 0 0 0 0 0
T_1	0 1 0 0 0 0
T_2	0 0 1 0 0 0
T_3	0 0 0 1 0 0
T_4	0 0 0 0 1 0
T_5	0 0 0 0 0 1

Table 11.4 State table for state generator.

Present state	START	Next state
T_0	0	T_0
T_0	1	T_1
T_1	d	T_2
T_2	d	T_3
T_3	d	T_4
T_4	d	T_5
T_5	d	T_0

The next-state equations for the state generator can be obtained directly from the state table because of the one-hot state assignment. From the table, it is seen that T_0 will be the next state if T_5 is the present state <u>OR</u> T_0 is the present state <u>AND</u> START = 0. Thus

$$Q_0^+ = Q_0 \bullet START' + Q_5$$

Similarly,

$$Q_1^+ = Q_0 \bullet START$$
$$Q_2^+ = Q_1$$
$$Q_3^+ = Q_2$$
$$Q_4^+ = Q_3$$
$$Q_5^+ = Q_4$$

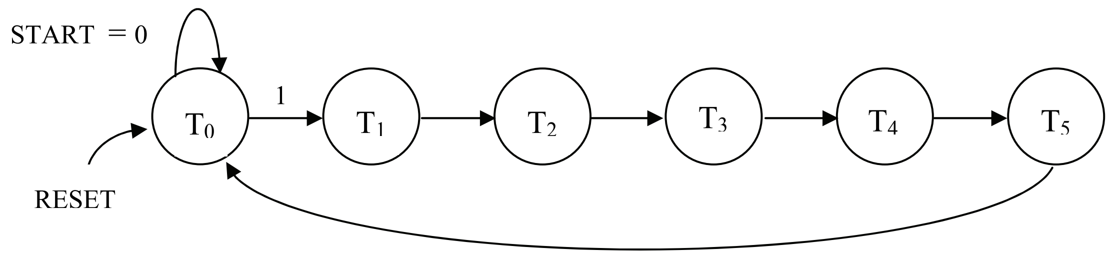


Figure 11.11 State diagram for state generator.

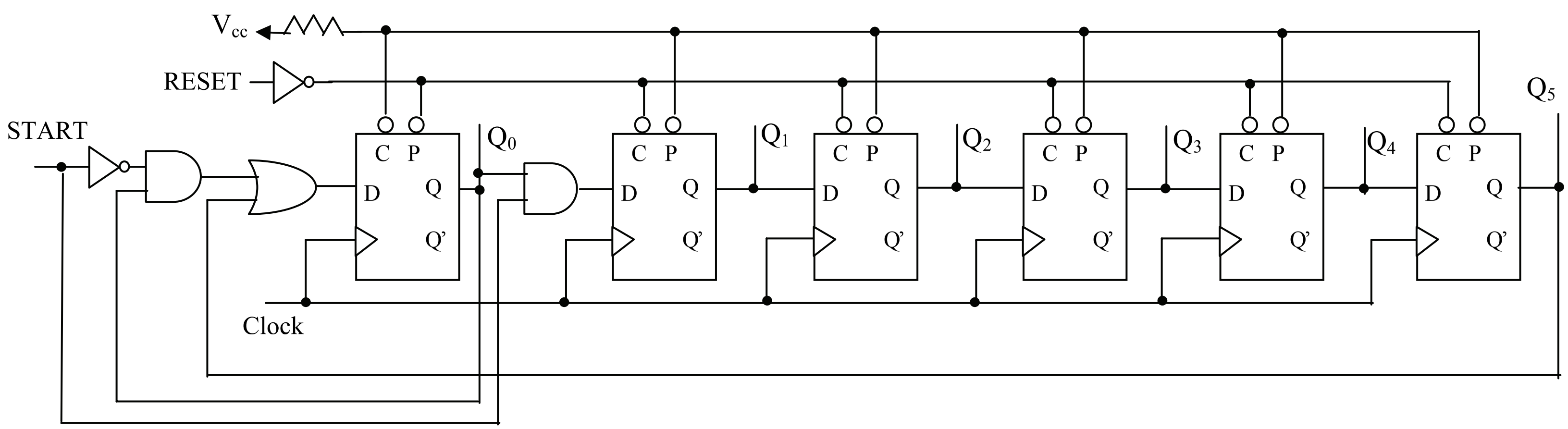


Figure 11.12 State generator.

The next-state equations can also be obtained from a state diagram. Figure 11.11 is a state diagram converted from the state table in Table 11.4. From the diagram, it is seen that there are two lines terminating at T_0. (RESET is for initialization and is treated separately.) A product term is derived from each line that terminates at T_0. One of the two lines originates from T_5. Therefore, T_0 is the next state if T_5 is the present state or if $Q_5 = 1$. The other line starts from T_0 and terminates at T_0 when START = 0. This means T_0 is the next state if T_0 is the present state and START = 0, or if Q_0•START' = 1. The result is the same as the next-state equation derived from the state table. Thus the next-state equation for a state can be obtained from all the lines that terminate at this state. The next-state equations for the other five states can be derived in a similar way.

When D flip-flops are used for the state generator, $Q_i^+ = D_i$, where i = 0, 1, ... ,5. A circuit diagram for the state generator is shown in Figure 11.12. To initialize the state generator to 100000, the inverted RESET input is connected to the asynchronous P (preset) of flip-flop 0 and to the asynchronous C (clears) of the other five flip-flops. All other asynchronous active-low presets and clears are de-asserted by assigning a value of 0. The state generator is initialized by applying a positive pulse at the RESET input.

Design of Control Circuit

Since the control signals generated in each state are not the same, the outputs of the control circuit are also functions of the states. Figure 11.13 is a block diagram for the control circuit. In designing the control circuit, it is noted that the functions of the shift registers are controlled by two selection signals s_1 and s_0. Hold1, Hold2, Load1, Load2, and Shift have to be converted to s_1 and s_0. Their relationships are listed in Table 11.5 for each asserted control signal. When Load and Shift are not asserted, the contents of a register should remain unchanged. Therefore s_1s_0 should be 00.

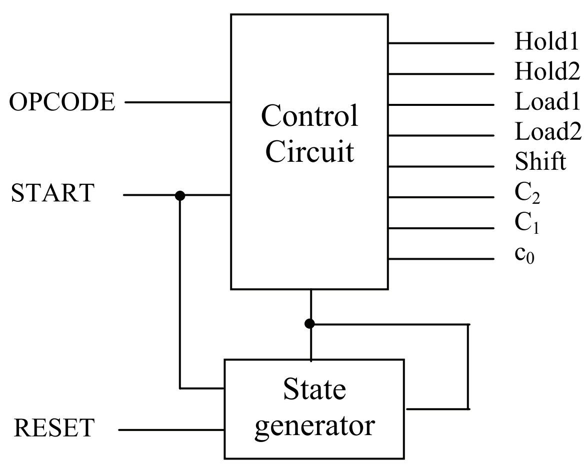


Figure 11.13 Block diagram for control circuit.

Table 11.5 Conversion of asserted signals to selection signals for shift register.

Asserted signal	$(s_1s_0)_{R1}$	$(s_1s_0)_{R2}$
Hold1	0 0	N/A
Hold2	N/A	0 0
Load1	1 1	N/A
Load2	N/A	1 1
Shift	0 1	0 1

The control circuit is a combinational circuit. Its outputs are functions of T_0, T_1, ..., T_5, OPCODE, and START. The truth table for the outputs of the control circuit in Table 11.6 can be obtained from the ASM chart in Figure 11.10. It is assumed that the control signals no longer depend on the value of START after the first operation has started. When the first number is loaded into T_0, it is also assumed that the contents of R_2 and the D flip-flop are not needed anymore. Therefore don't-care values are assigned to s_1s_0 of R_2, C_2, C_1, and c_0.

Table 11.6 Truth table for the control circuit.

State	START	$(s_1s_0)_{R1}$	$(s_1s_0)_{R2}$	C_2	C_1	c_0
T_0	0	0 0	0 0	0	1	d
T_0	1	1 1	d d	d	d	d
T_1	d	0 0	1 1	1	0	OPCODE
T_2	d	0 1	0 1	1	1	d
T_3	d	0 1	0 1	1	1	d
T_4	d	0 1	0 1	1	1	d
T_5	d	0 1	0 1	1	1	d

A sum-of-products expression for each control signal can be obtained directly from the truth table by examining the values of this control signal that are 1. Each value of 1 corresponds to one product term. Thus

$$(s_1)_{R1} = T_0 \bullet START$$

$$(s_0)_{R1} = T_0 \bullet START + T_2 + T_3 + T_4 + T_5$$

$$(s_1)_{R2} = T_1$$

$$(s_0)_{R2} = T_1 + T_2 + T_3 + T_4 + T_5$$

$$C_2 = T_1 + T_2 + T_3 + T_4 + T_5$$

$C_1 = T_0 + T_2 + T_3 + T_4 + T_5$

$c_0 = \text{OPCODE}$

To obtain a simple expression for s_1 of R_2, the don't-care value takes on a value of 0, so that the expression is independent of T_0. The don't-care values for s_1 of R_2 and C_2 are also given a value of 0 for the same reason. However, the don't-care value for C_1 takes on a value of 1. A value of 0 will change the term T_0 in the expression for C_1 to T_0•START'. All don't-care values for c_0 are specified as OPCODE so that c_0 does not require any gate for implementation.

Sometimes, a simpler expression can be obtained by examining the complement of a control signal. For instance, the selection signal s_0 for both registers and C_2, C_1 are re-examined using Table 11.7.

Table 11.7 Truth table for s_0', C_2', and C_1'

State	START	$(s_0')_{R1}$	$(s_0')_{R2}$	C_2'	C_1'
T_0	0	1	1	1	0
T_0	1	0	d	d	d
T_1	d	1	0	0	1
T_2	d	0	0	0	0
T_3	d	0	0	0	0
T_4	d	0	0	0	0
T_5	d	0	0	0	0

From Table 11.7, $(s_0')_{R1} = T_0 \bullet \text{START}' + T_1$

$(s_0')_{R2} = T_0$

$C_2' = T_0$

$C_1' = T_1$

By complementing each of the above equations, they become

$(s_0)_{R1} = (T_0 \bullet \text{START}' + T_1)'$

$(s_0)_{R2} = T_0'$

$C_2 = T_0'$

$C_1 = T_1'$

These expressions are simpler than the previously derived expressions. In fact, some of the simpler expressions can be obtained directly from the more complex ones. For instance, $(T_1 + T_2 + T_3 + T_4 + T_5)$ implies that if the state generator is in one of the five states T_1, T_2, T_3, T_4, or T_5, then it is not in T_0. Thus $T_1 + T_2 + T_3 + T_4 + T_5 = T_0$'. If the state generator is in T_0, then it is NOT in T_1, OR T_2, OR T_3, OR T_4, OR T_5 and $T_0 = (T_1 + T_2 + T_3 + T_4 + T_5)$'.

11.6 Revisit of Arithmetic Processor

The data path in Figure 11.9 is modified in Figure 11.14 in order to perform more arithmetic functions. Instead of OPCODE, the modified data path has three operation codes OP_2, OP_1, and OP_0. The initial carry of an arithmetic operation is OP_0. The input circuit is replaced with a 4-to-1 multiplexer. The four data inputs to the multiplexer are b_i, b_i', a_i, and 1. OP_2 and OP_1 are the control signals of the multiplexer. Since there is no need to determine the initial carry, the decision box and conditional outputs for the initial carry are removed from the ASM chart in Figure 11.10, which is shown in Figure 11.15. The processor can perform eight different functions defined by OP_2 OP_1 OP_0.

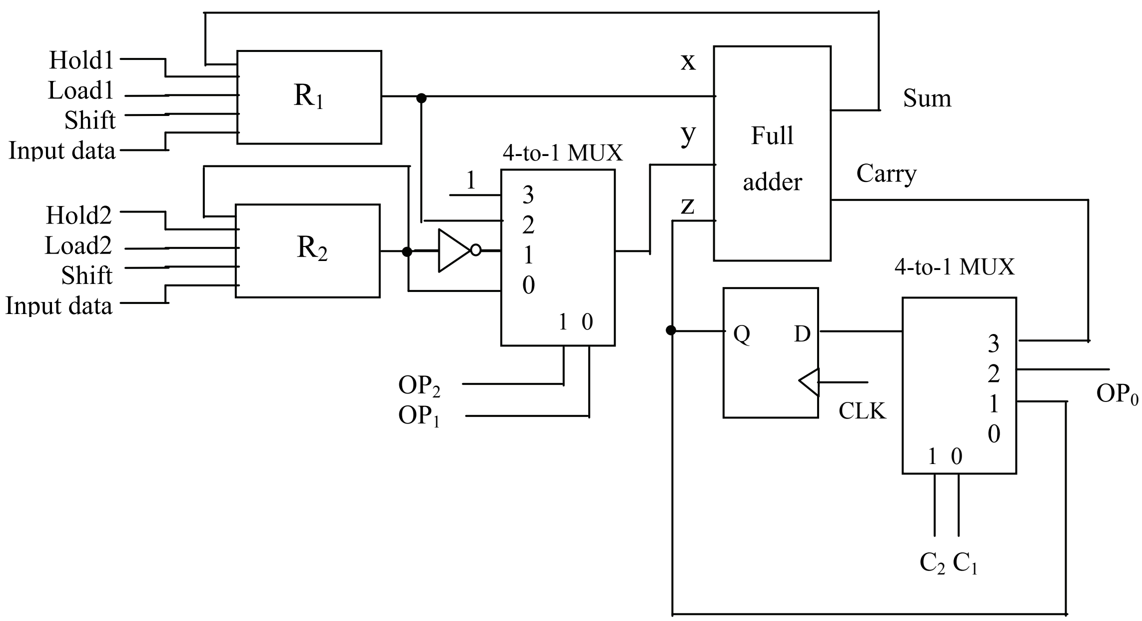


Figure 11.14 A processor for eight arithmetic functions.

In the last four clock cycles of an arithmetic operation, the full adder performs the following addition:

$$x_3x_2x_1x_0 + y_3y_2y_1y_0 + c_0$$

x_i and y_i are the x and y inputs to the full adder in state T_{i+2}. Since R_1 is connected to x directly, $x_3x_2x_1x_0 = a_3a_2a_1a_0$. $y_3y_2y_1y_0$ depends on OP_2 and OP_1. The eight different functions performed by the processor are analyzed and listed in Table 11.8. Note that

$b'_3b'_2b'_1b'_0$, the 1's complement of B, is equivalent to $(-B-1)$. If $y_i = 1$, $y_3y_2y_1y_0 = 1111$, which is -1 in decimal.

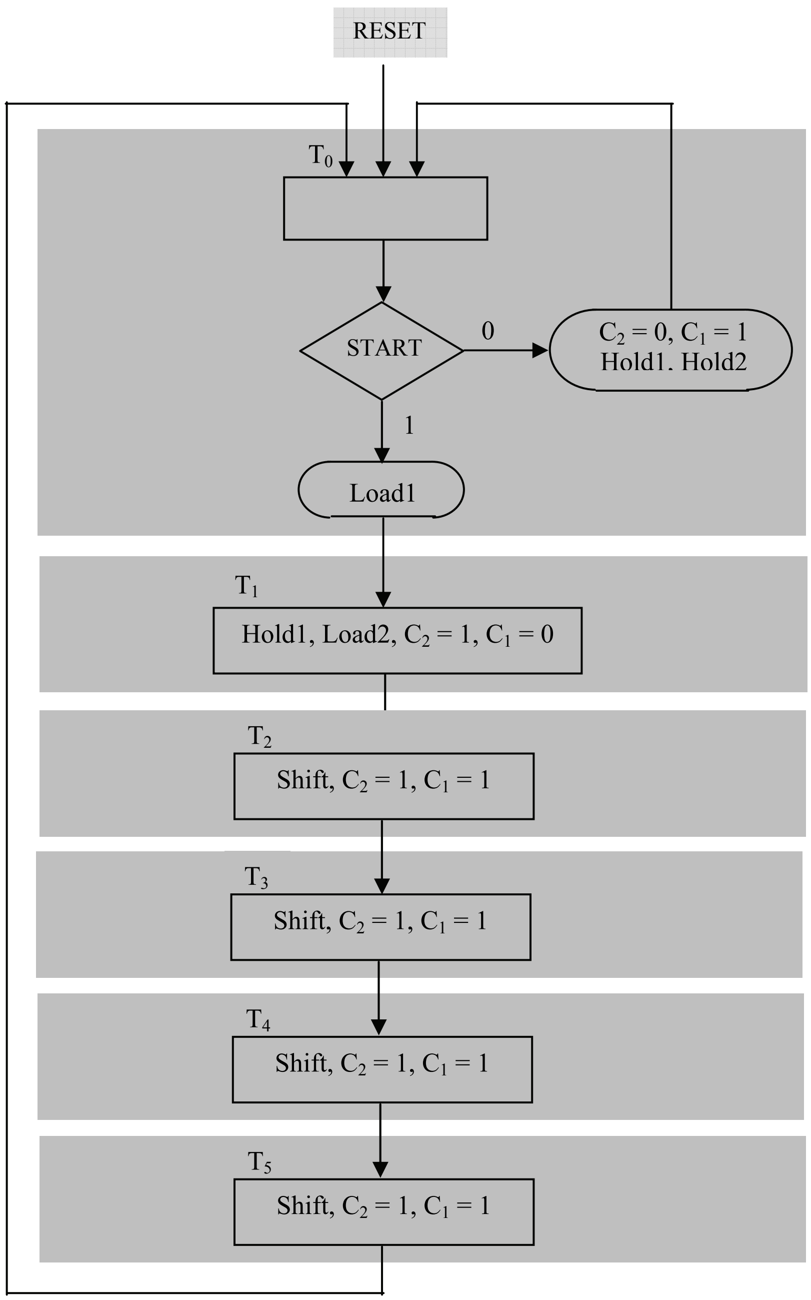


Figure 11.15 ASM chart for the arithmetic processor in Figure 11.14.

Table 11.8 Arithmetic functions for the processor in Figure 11.14.

OP_2 OP_1 OP_0	$x_3x_2x_1x_0 + y_3y_2y_1y_0 + c_0$	Arithmetic function
0 0 0	$a_3a_2a_1a_0 + b_3b_2b_1b_0 + 0$	A + B
0 0 1	$a_3a_2a_1a_0 + b_3b_2b_1b_0 + 1$	A + B + 1
0 1 0	$a_3a_2a_1a_0 + b_3'b_2'b_1'b_0' + 0$	A – B – 1
0 1 1	$a_3a_2a_1a_0 + b_3'b_2'b_1'b_0' + 1$	A – B
1 0 0	$a_3a_2a_1a_0 + a_3a_2a_1a_0 + 0$	2A
1 0 1	$a_3a_2a_1a_0 + a_3a_2a_1a_0 + 1$	2A + 1
1 1 0	$a_3a_2a_1a_0 + 1111 + 0$	A – 1
1 1 1	$a_3a_2a_1a_0 + 1111 + 1$	A

PROBLEMS

1. Find the two's complement for each of the following 8-bit numbers.

(a)	00100101	(b)	11100000
(c)	11111111	(d)	00010010
(e)	10000011	(f)	01111111

2. Find the decimal equivalent for each of the following 8-bit signed numbers.

(a)	11101001	(b)	01100000
(c)	11111110	(d)	10010110
(e)	10000000	(f)	01111111

3. Convert each of the following decimal numbers to a 12-bit signed numbers.

(a)	+2025	(b)	+850
(c)	–1753	(d)	–2047
(e)	–753	(f)	–278

4. Given below are four pairs of 8-bit numbers. The first number is A and the second B. Calculate A + B, A – B, –A + B, –A – B for each of the four pairs of numbers using two's complement arithmetic. Verify your results by decimal arithmetic. If the results are not the same, explain why.

(a)	01010101, 00001010	(b)	01101011, 00101010
(c)	11101010, 00101111	(d)	10000000, 01111111

5. Convert the state diagram in Figure P11.1 to an ASM chart.

6. Convert the state diagram in Figure P11.2 to an ASM chart.

Figure P.11.1

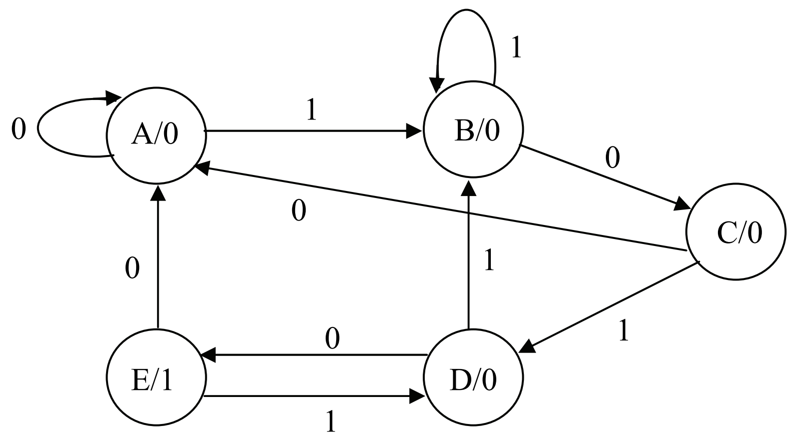
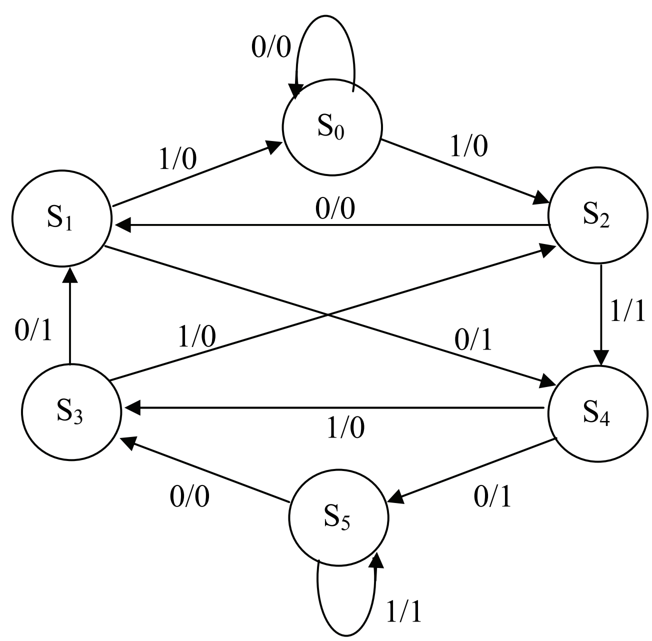


Figure P.11.2



7. Given in Table P11.1 is the output table of a control circuit. The circuit has one input X and four outputs y_3, y_2, y_1, and y_0. The states are represented by a 4-bit ring counter. Implement the control circuit using a minimum number of gates.

8. Realize the state generator specified by the state assignment table in Tables P11.3 and the state diagram in Figure P11.3. The state generator is controlled by an external input x. RESET is an external active-high input. The state generator is reset to T_0 when RESET is asserted.

9. The operations of the data-path in Figure P11.4 is described by the ASM chart in Figure 11.14 and depend on the operation code $Op_2Op_1Op_0$. Determine the eight arithmetic functions performed by the data-path.

10. The operations of the data-path in Figure P11.5 is described by the ASM chart in Figure 11.14 and depend on the operation code $Op_4Op_3Op_2Op_1Op_0$. Determine the values of $Op_4Op_3Op_2Op_1Op_0$ for the following arithmetic functions. Note that the result is stored in R_1.

(a)	$A - 1$	(b)	$-B$
(c)	$A - B - 1$	(d)	B
(e)	$-A - B - 1$	(f)	$2A + 1$
(g)	$A - B$	(h)	0 (Clear R_1)

Table P11.1

State	X	$y_3\ y_2\ y_1\ y_0$
T_0	0	d d d d
T_0	1	1 0 0 1
T_1	0	1 1 0 0
T_1	1	1 0 0 1
T_2	0	1 0 1 0
T_2	1	1 0 0 1
T_3	0	0 1 1 1
T_3	1	0 0 1 1

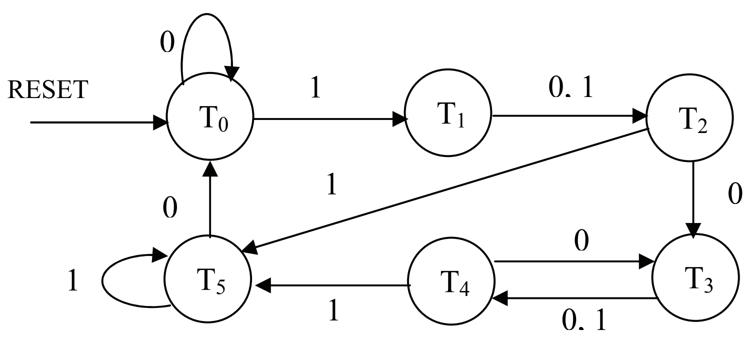


Figure P11.3

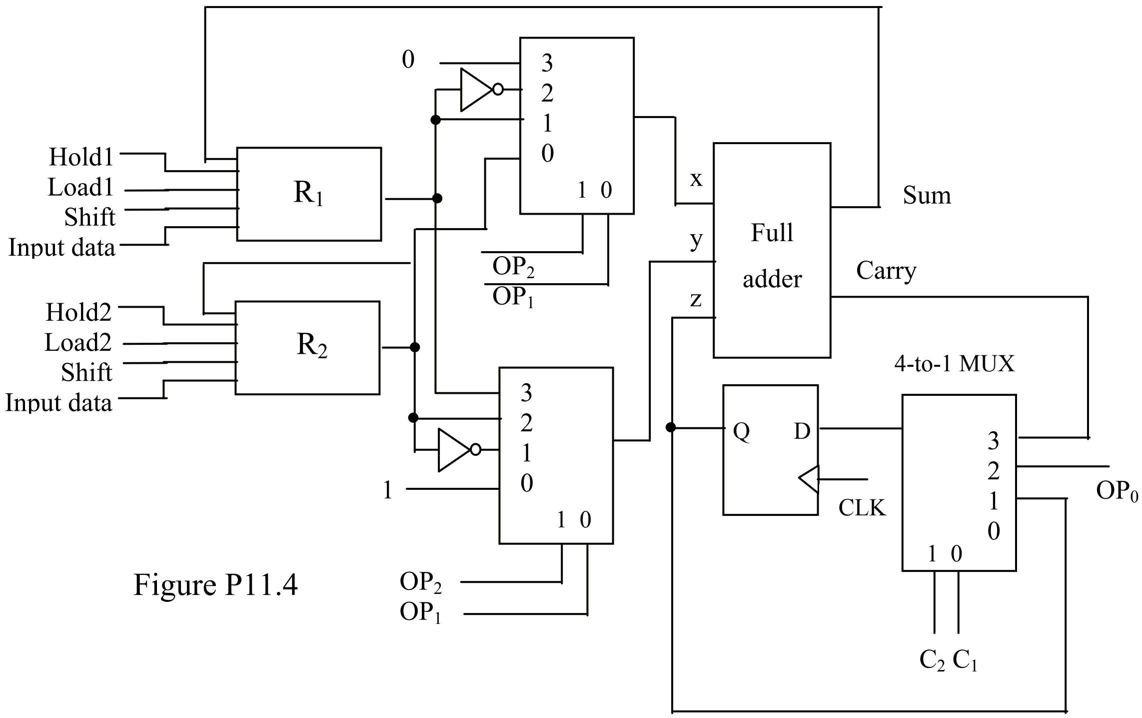


Figure P11.4

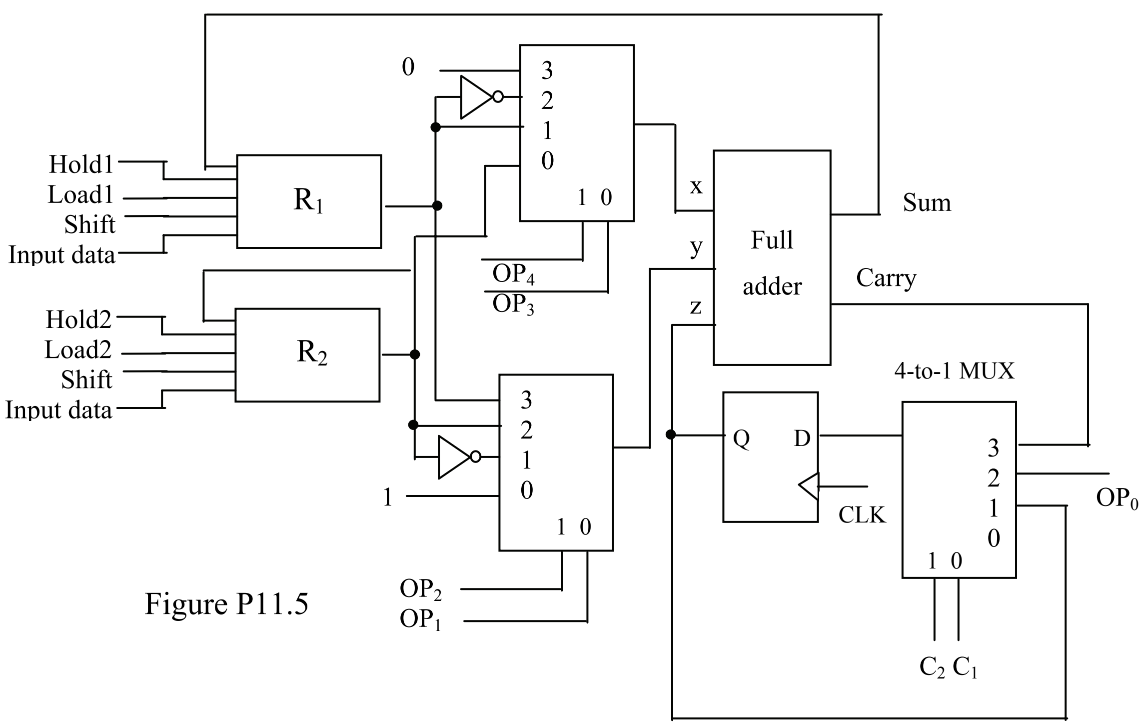


Figure P11.5