Datapath Design

Circuit Arrangement for Restoring Binary Division (Positive Numbers)

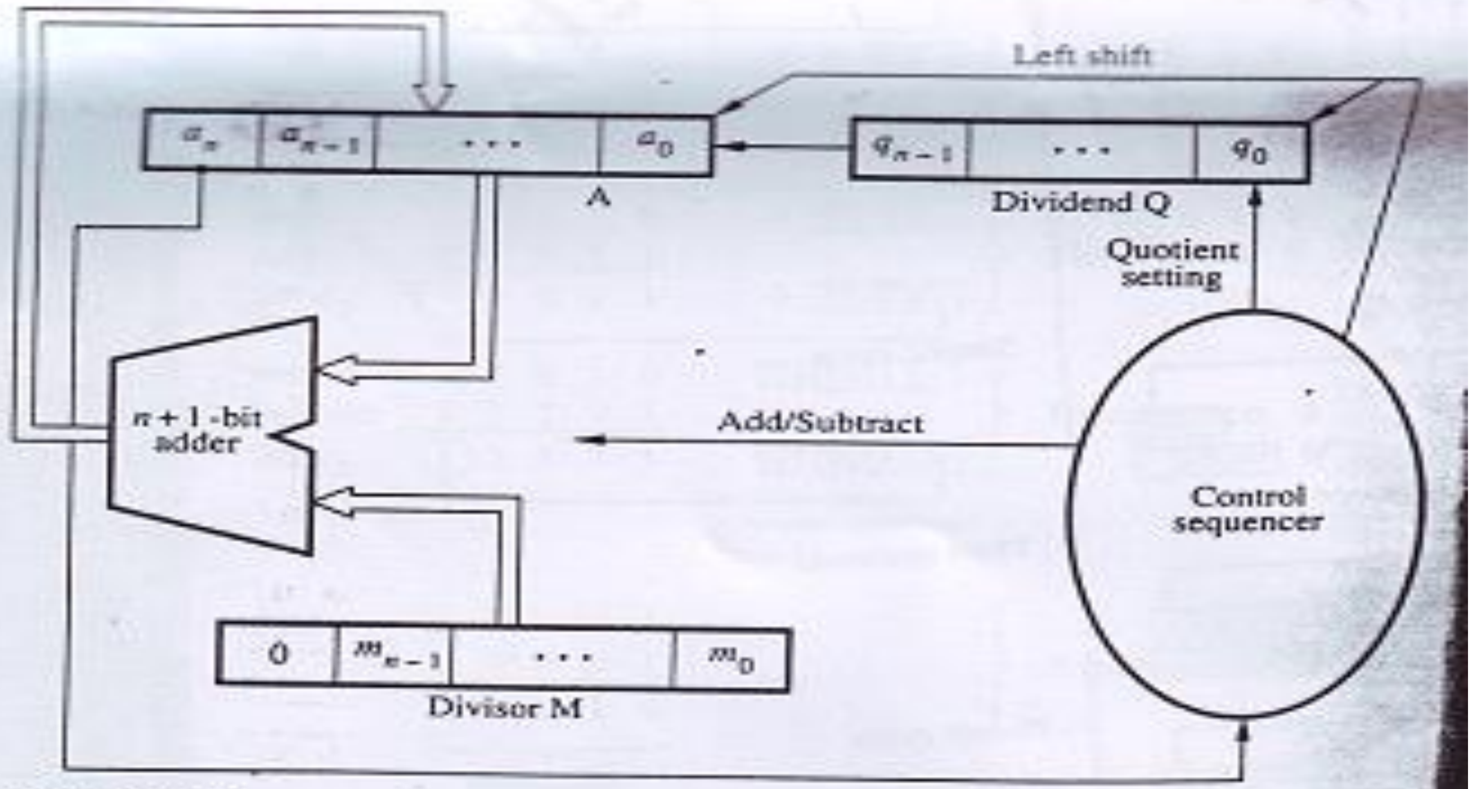


FIGURE 6.20
Circuit arrangement for binary division.

Restoring Division

1. Do the following *n* times:

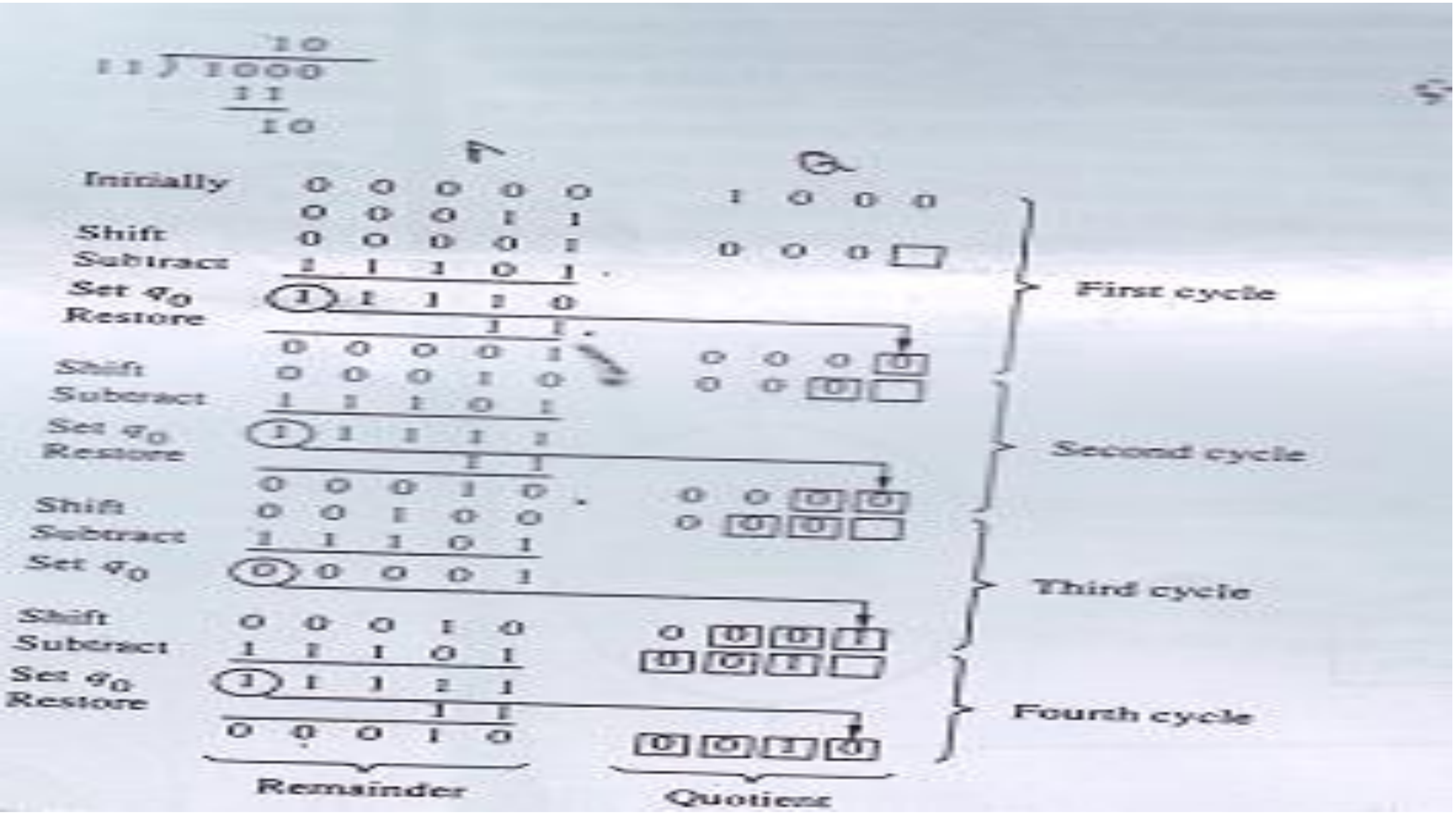
   (a) Shift *A* and *Q* left one binary position.

   (b) Subtract *M* from *A* and place the result back in *A*.

   (c) If the sign of *A* is 1, set q_0 to 0 and add *M* back to *A*; otherwise, set q_0 to 1.

Note: Subtraction is performed by using 2’s complement arithmetic.

Example



```
      10
11 ) 1000
      11
      10
```

	A	Q	
Initially	0 0 0 0 0	1 0 0 0	
	0 0 0 1 1		
Shift	0 0 0 0 1	0 0 0 □	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0		First cycle
Restore	1 1		
	0 0 0 0 1	0 0 0 0	
Shift	0 0 0 1 0	0 0 0 □	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1		Second cycle
Restore	1 1		
	0 0 0 1 0	0 0 0 0	
Shift	0 0 1 0 0	0 0 0 □	
Subtract	1 1 1 0 1		
Set q_0	0 0 0 0 1	0 0 0 1	Third cycle
Shift	0 0 0 1 0	0 0 1 □	
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1		Fourth cycle
Restore	1 1		
	0 0 0 1 0	0 0 1 0	
	Remainder	Quotient	

5

Non-restoring Division

- Restoring division algorithm can be improved by avoiding the need for restoring *A* after an unsuccessful subtraction.
- If A is positive, we shift left and subtract *M*. That is we perform *2A-M*.
- If A is negative, we restore it by performing A+M and then we shift it left and subtract M. This is equivalent to 2A+M.

Algorithm for non-restoring Division

Step 1:

Do the following *n* times:

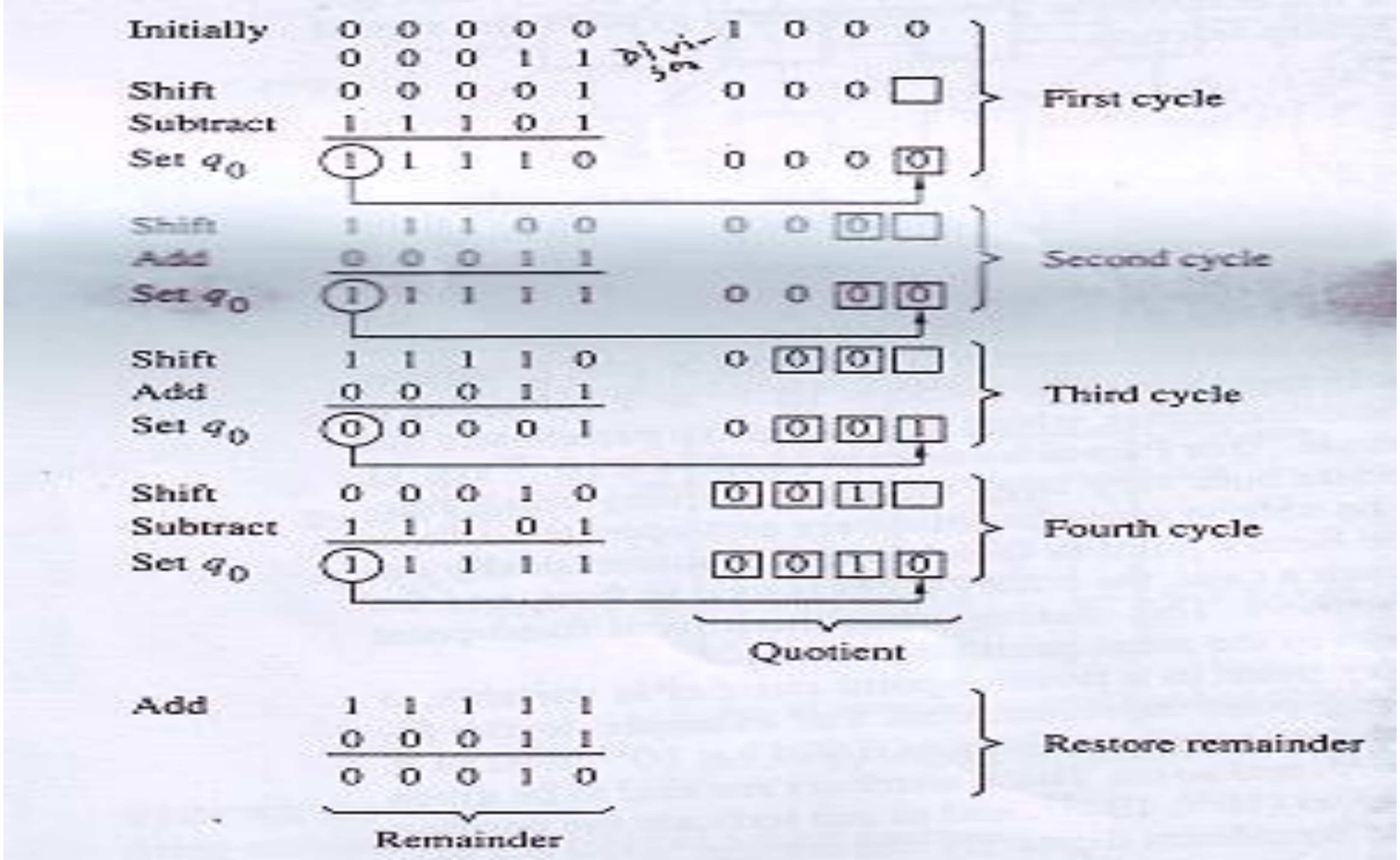
if sign of *A* is 0, shift *A* and *Q* left one bit position and subtract *M* from *A*; otherwise shift *A* and *Q* left and add *M* to *A*.

if the sign of *A* is 0, set q_0 to 1; otherwise set q_0 to 0.

Step 2:

if sign of *A* is 1, add *M* to *A*.

Example



Initially	0 0 0 0 0	1 0 0 0	
	0 0 0 1 1 Divi-sor		
Shift	0 0 0 0 1	0 0 0 □	First cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 0	0 0 0 0	
Shift	1 1 1 0 0	0 0 0 □	Second cycle
Add	0 0 0 1 1		
Set q_0	1 1 1 1 1	0 0 0 0	
Shift	1 1 1 1 0	0 0 0 □	Third cycle
Add	0 0 0 1 1		
Set q_0	0 0 0 0 1	0 0 0 1	
Shift	0 0 0 1 0	0 0 1 □	Fourth cycle
Subtract	1 1 1 0 1		
Set q_0	1 1 1 1 1	0 0 1 0	
		Quotient	
Add	1 1 1 1 1		Restore remainder
	0 0 0 1 1		
	0 0 0 1 0		
	Remainder		

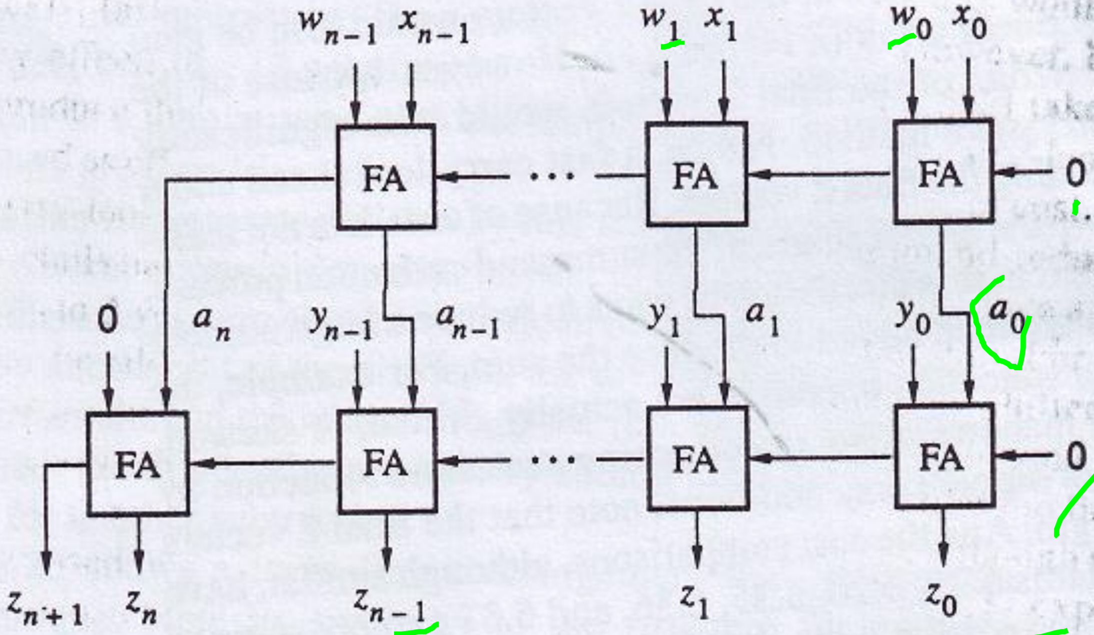
Signed Division

- Negate the quotient if the sign of the operands are opposite.
- Sign of the non-zero remainder should match the dividend.
- Example:
  - +7 / +2 : Q= 3 and R=1
  - -7 / +2 : Q= -3 and R=-1
  - -7 / -2 : Q = 3 and R = -1
  - +7 / -2: Q = -3 and R = 1

Carry Save Addition

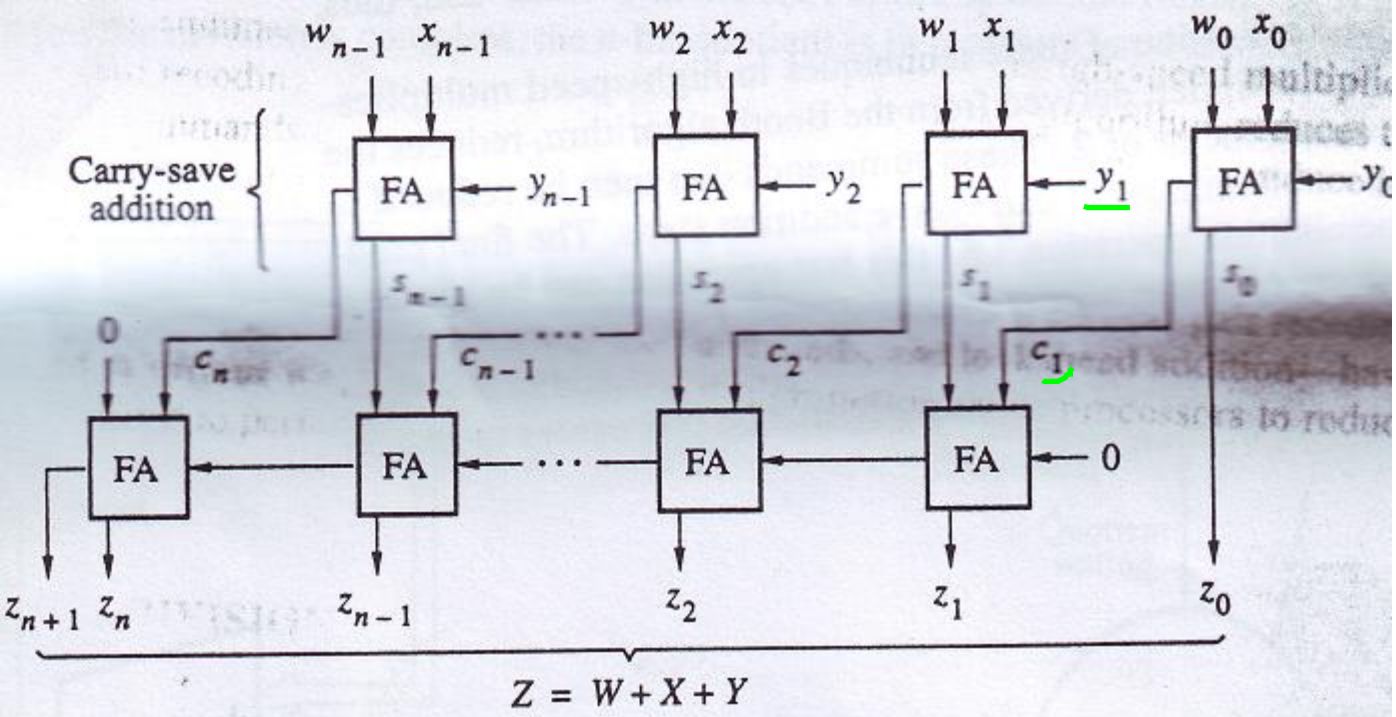
- ✔ An n-bit carry save adder consists of n disjoint full adders. It’s input is three n-bit numbers to be added and the output consists of the n-sum bits forming a word *S* and the n carry bits forming a word *C*.
- ✔ Carry connections are shifted to the left correspond to normal carry propagation.
- ✔ To obtain the final result, *S* and *C* must be added by a conventional adder with carry propagation.

Carry Save Addition



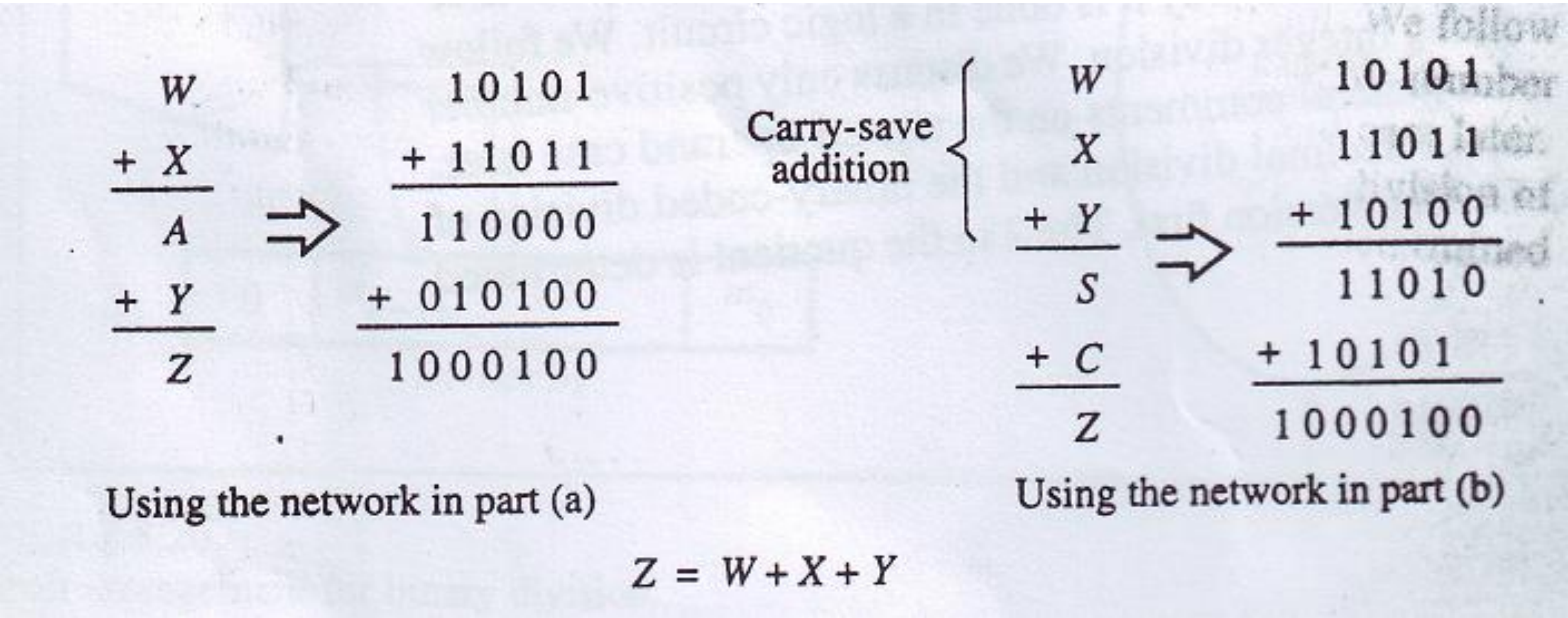
(a) $W + X + Y$ using two ripple-carry adders

Carry Save Addition



(b) $W + X + Y$ using carry-save addition

Carry Save Addition



$$\begin{array}{r} W \\ +\ X \\ \hline A \\ +\ Y \\ \hline Z \end{array} \quad \Rightarrow \quad \begin{array}{r} 10101 \\ +\ 11011 \\ \hline 110000 \\ +\ 010100 \\ \hline 1000100 \end{array}$$

Using the network in part (a)

$$\text{Carry-save addition}\left\{\begin{array}{r} W \\ X \\ +\ Y \end{array}\right. \quad \begin{array}{r} \\ \\ \\ \hline S \\ +\ C \\ \hline Z \end{array} \quad \Rightarrow \quad \begin{array}{r} 10101 \\ 11011 \\ +\ 10100 \\ \hline 11010 \\ +\ 10101 \\ \hline 1000100 \end{array}$$

Using the network in part (b)

$$Z = W + X + Y$$

Carry Save Addition

Advantages:

- ✔ All bits of *S* and *C* vectors are produced in a short, fixed amount of time.
- ✔ Carry propagation takes place only in the second row.
- ✔ Since all bits of *S* and *C* are available in parallel, a carry-lookahead adder can be used effectively to add the *S* and *C*.