Lecture # 3 (Final)
Processor Logic Design (1st Part)
ALU Design

Department of EEE

Introduction

- A processor unit is that part of a digital system that implements the operations in the system.
- It comprises a number of registers and the digital functions that implement arithmetic, logic, shift, and transfer micro-operations.
- The processor unit, when combined with a control unit that supervises the sequence of micro-operations, is called a central processing unit (CPU).
- Central Processing Unit (CPU) consists of Memory (Register Banks), Arithmetic Logic Unit (ALU) and Control Unit (CU).
- There are **three organizations** for organizing a general-purpose processor unit:
  - Bus organization
  - Scratchpad memory
  - Two-port memory

AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

Bus Organization

- A bus organization has 4 registers, each register is connected to 2 multiplexers to form 2 buses A and B.
- To perform the **micro-operation**:
  - R1 = R2 + R3
- The control unit must provide binary selection variables to the following selector inputs –

1. MUX A Selector: to place contents of R2 onto bus A
2. MUX B Selector: to place contents of R3 onto bus B
3. ALU Function Selector: to provide the arithmetic operation A + B
4. Shift Selector: for direct transfer from the output of the ALU onto output bus S (no shift)
5. Decoder Destination Selector: to transfer the contents of bus S into R1

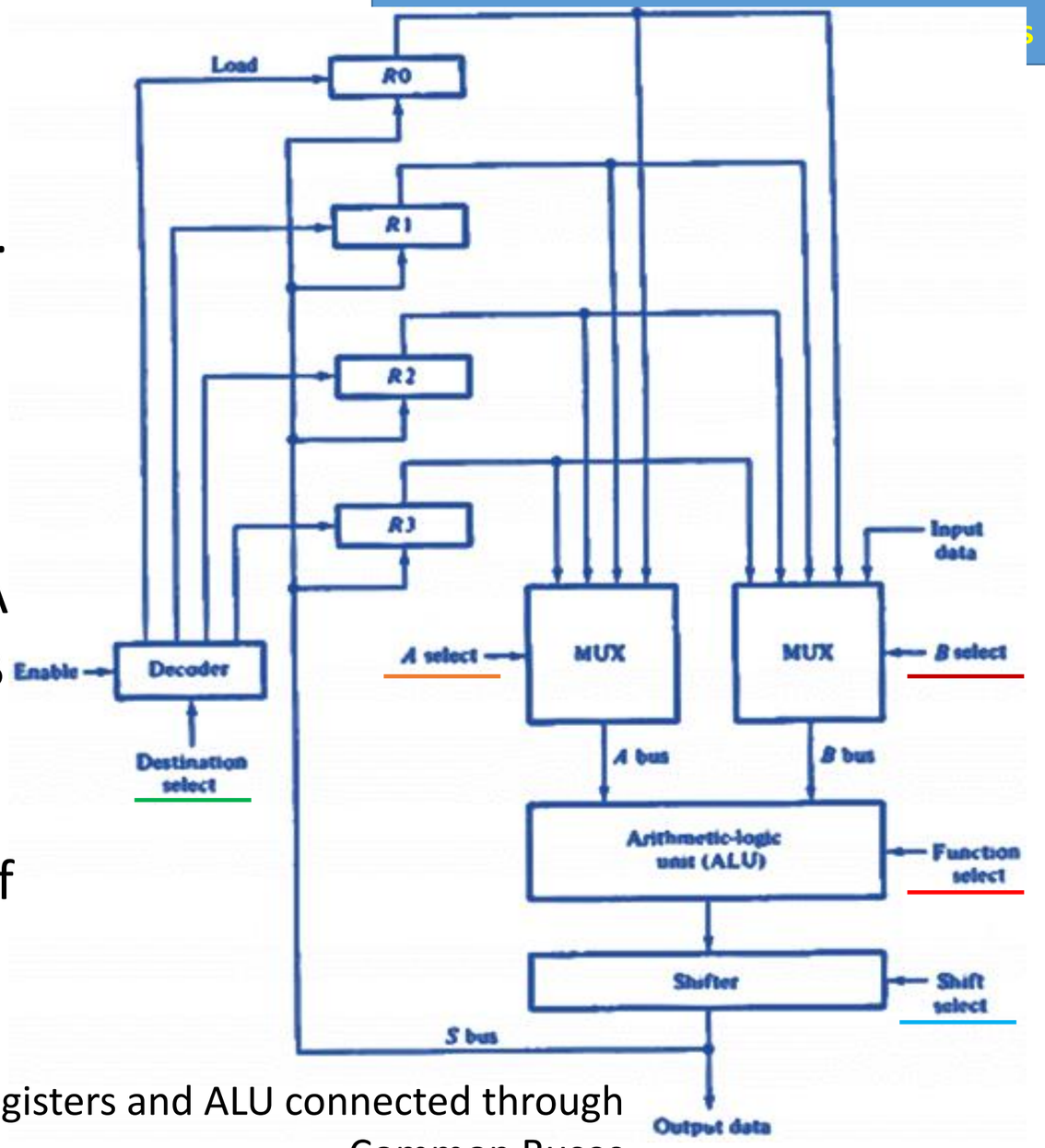


Fig. 9.1 Processor registers and ALU connected through Common Buses

Bus Organization

- The **five control selection variables must be generated** simultaneously and must be available during one common clock pulse interval.
- The binary information from two source registers propagates through the combinational gates in the multiplexers, the ALU, and the shifter, to the output bus and into the inputs of the destination register all during one clock pulse interval.
- Then, when the next clock pulse arrives, the binary information on the output bus is transferred to R1.

Scratchpad Memory

- The registers in a processor unit can be enclosed within a small memory unit. When included in a processor unit, a small memory is sometimes called a **scratchpad memory**.
- The use of a small memory is a cheaper alternative to connecting processor registers through a bus system. The difference between the two systems is the manner in which information is selected for transfer into the ALU.
- In the bus system, the information transfer is selected by the multiplexers that form the buses.
- On the other hand, a single register in a group of registers organized as a small memory must be selected by means of an address to the memory unit.

Scratchpad Memory

- To perform the operation R1 = R2 + R3, the control must provide binary selection variables to perform the following sequence of three microoperations:
  - T1: A ← M[010] Read R2 into register A
  - T2: B ← M[011] Read R3 into register B
  - T3: M [001] ← A+B Perform the operation in ALU and transfer the result to R1
- The reason for a sequence of 3 microoperations, instead of just 1 as in a bus-organized processor, is due to the limitation of the memory unit having only one set of address terminals but two source registers are to be accessed. So, multiple memory addresses can't be accessed simultaneously.

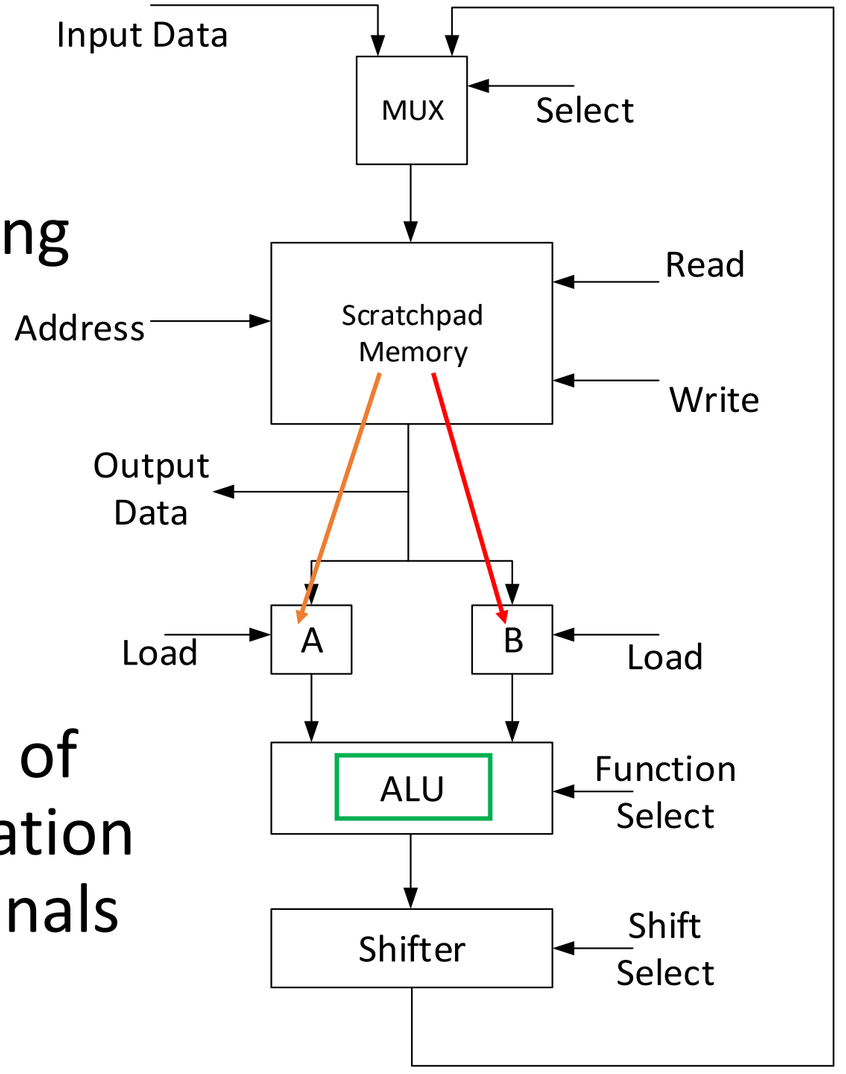


Fig. 9.2 Processor unit employing a scratchpad memory

Processor Unit with 2-Port Memory

- Some processors employ a 2-port memory to overcome the delay caused when reading two source registers.
- If the destination register is the same as one of the source registers, then the entire micro-operations can be done within one clock pulse period.

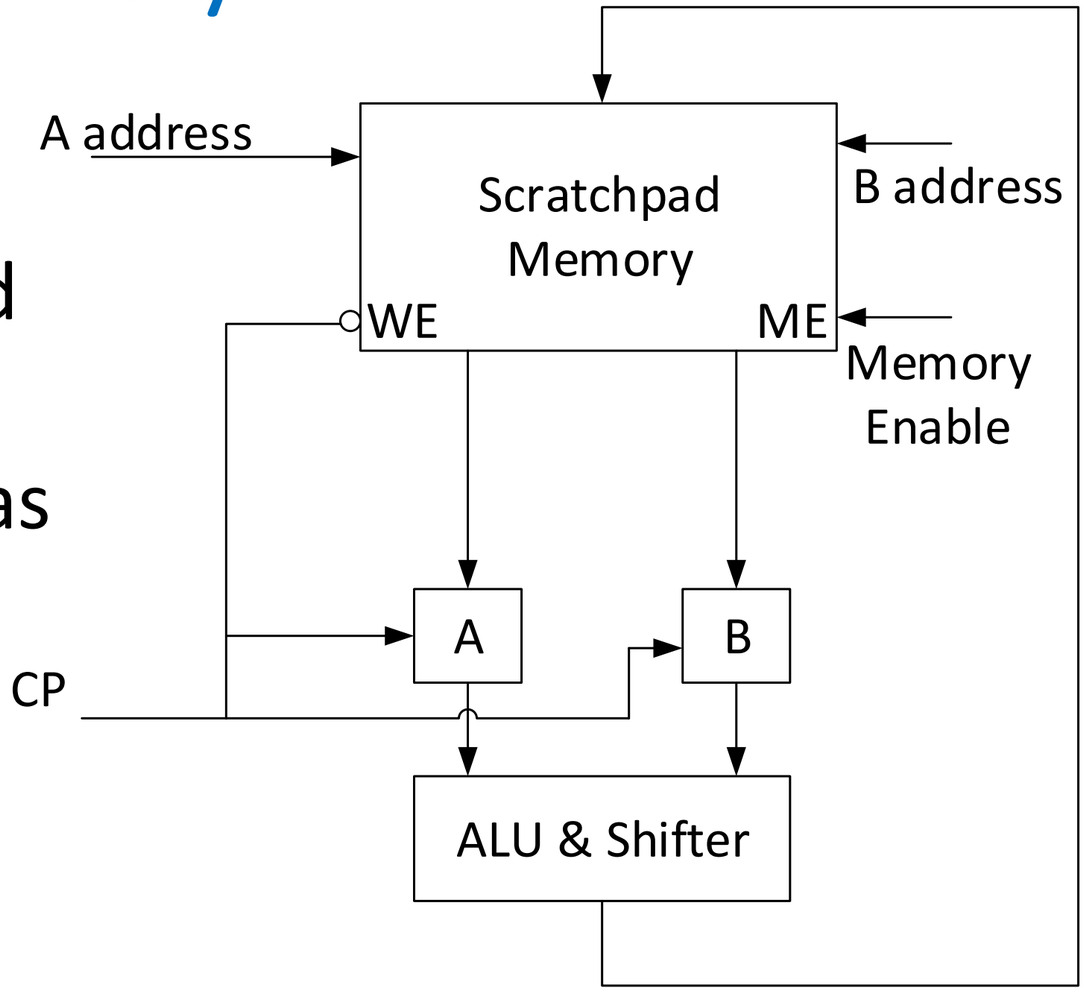


Fig. 9.3 Processor unit with a 2-port memory

Processor Unit with 2-Port Memory

- When CP = 1, **A and B latches** are open and accept information coming from memory. The WE is also in a HIGH state and disables write operation and enables read operation. Thus, when CP = 1, words selected by A and B addresses are read from memory and placed in A and B registers.
- When CP = 0, the latches are closed and retain the last data entered. ALU operation is performed and if a write is enabled since CP = 0, and also ME input is enabled, the result of the micro-operation is written into memory word defined by B address.

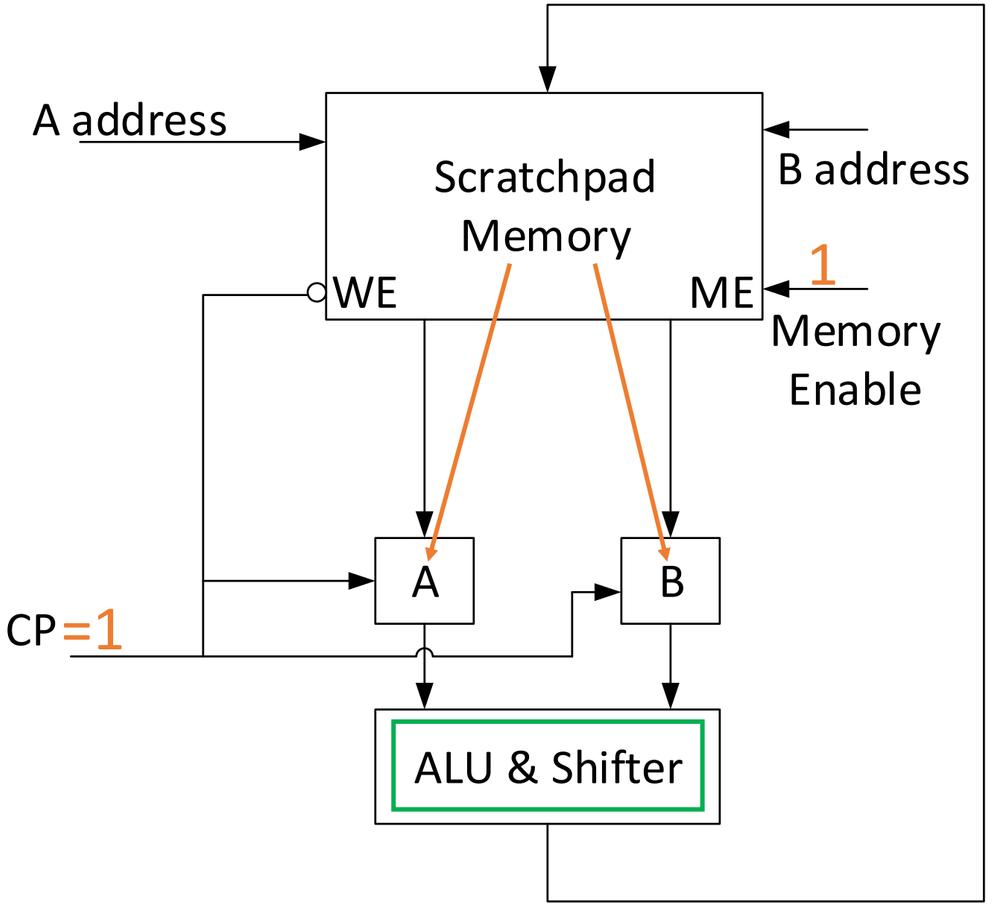
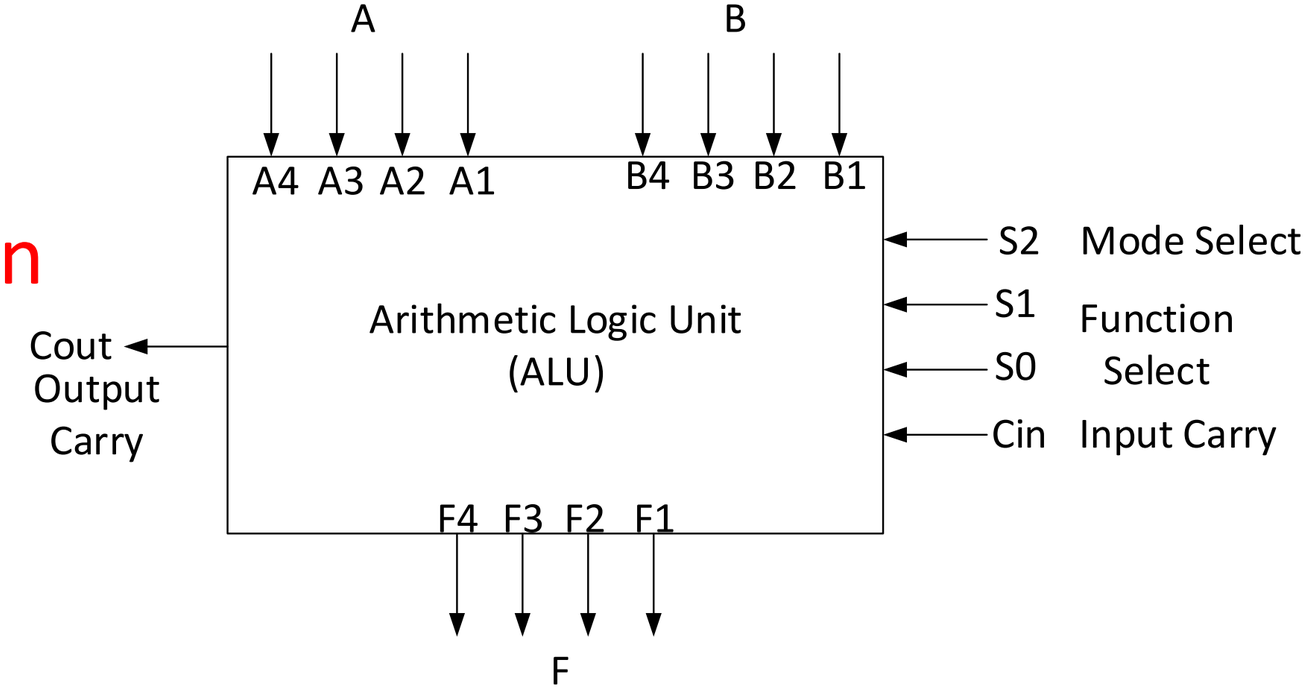


Fig. 9.3 Processor unit with a 2-port memory

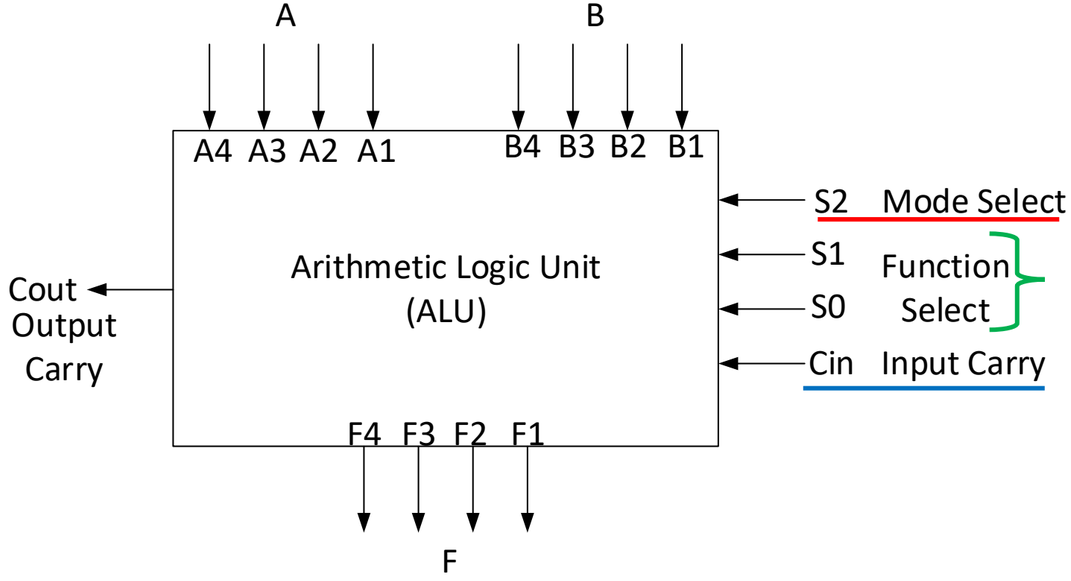
Arithmetic Logic Unit (ALU)

- An Arithmetic Logic Unit (ALU) is a multi-operation, combinational logic digital functional circuit.
- The ALU can perform a set of basic arithmetic and logic operations.
- The ALU has several selection lines to select a particular operation in the unit.
- The selection lines are decoded within the ALU so that *k* selection variables can specify up to 2^k distinct operations.



Arithmetic Logic Unit (ALU)

1. The mode select **s2** distinguishes between arithmetic and logic operations.
2. The two function-select inputs **s1 and s0** specify the arithmetic and logic operation to be generated.
3. The input and output carry (C_{in} and C_{out}) have meanings only during arithmetic operations.
4. The input carry **C_{in}** in the least significant position of an ALU is quite often used as the fourth selection variable that can double the number of arithmetic operations. Thus, a total of 8 arithmetic operations and 4 logical operations can be performed.

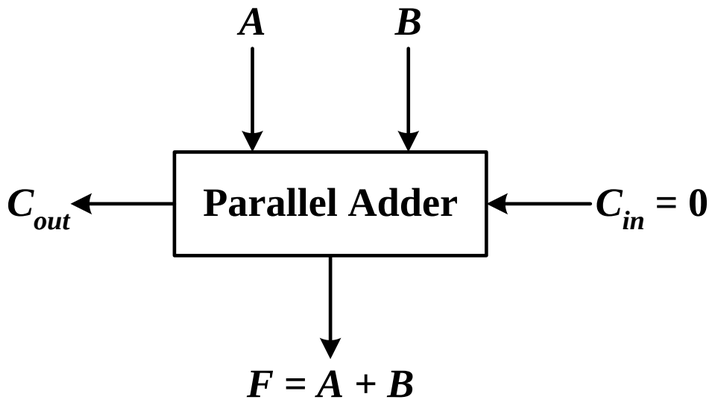


Design of Arithmetic Circuit

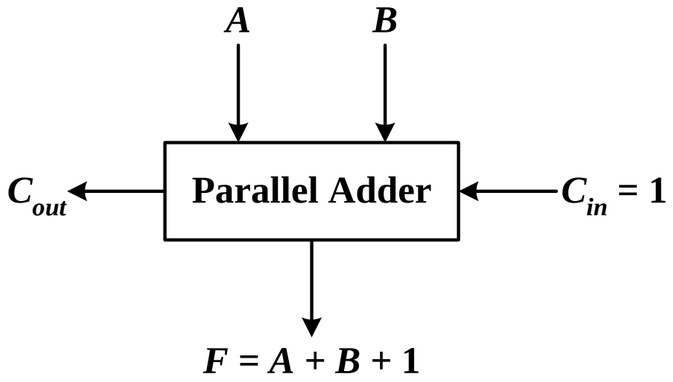
- The basic component of the arithmetic section of an ALU is the parallel adder.
- A parallel adder is constructed with several full adder circuits cascaded.
- By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations.

Operations obtained by controlling one set of inputs to a parallel adder

$S_1S_0 = 01$

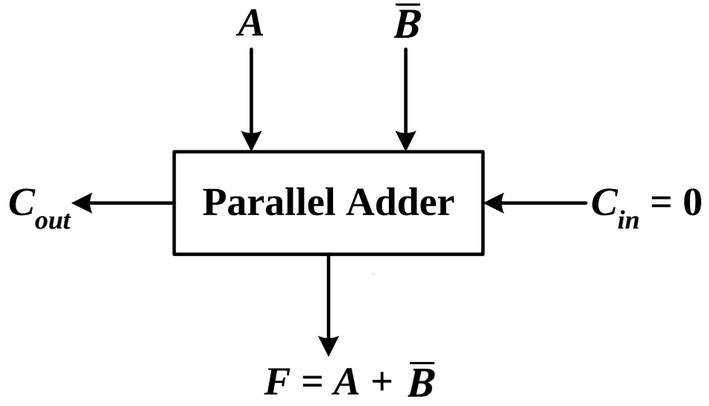


(a) Addition without input carry

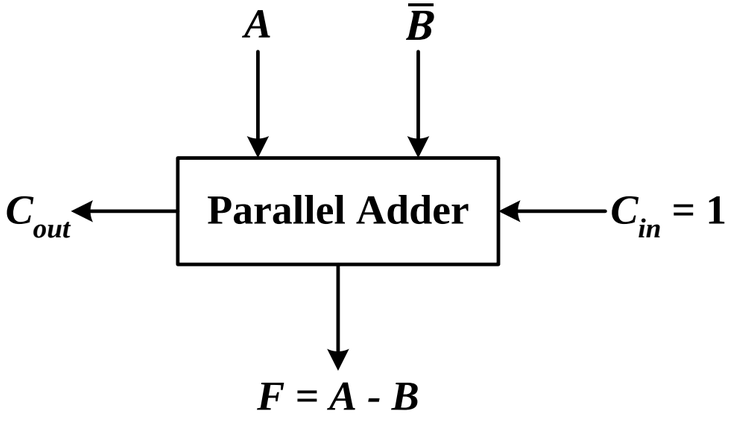


(b) Addition with input carry

$S_1S_0 = 10$



(c) Addition with 1's complement of B

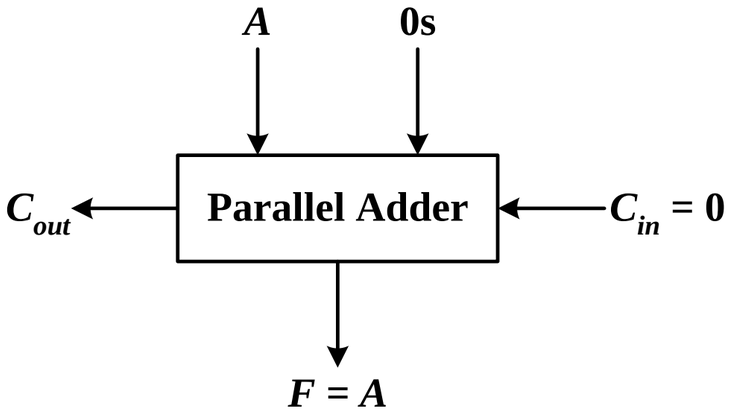


(d) Subtraction

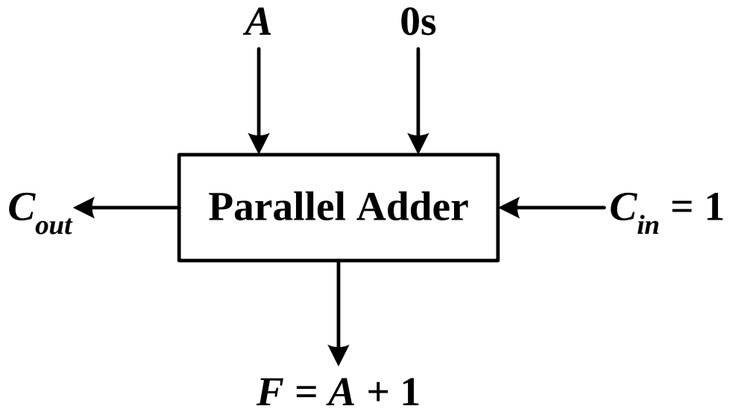
Fig. 9.6

Operations obtained by controlling one set of inputs to a parallel adder

S_1S_0 = 00

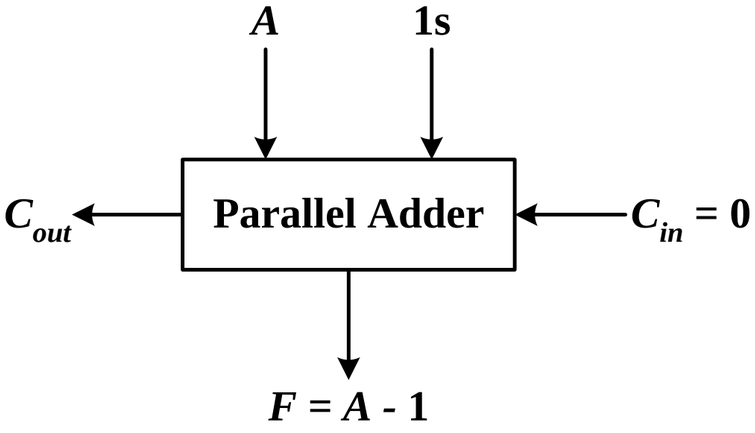


(e) Transfer of *A*

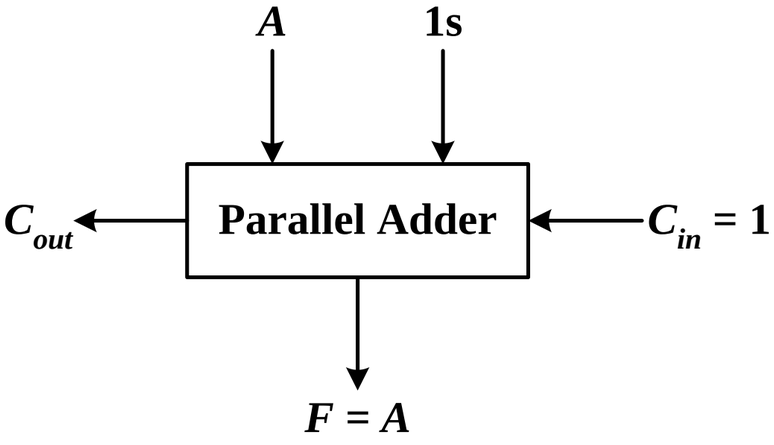


(f) Increment of *A*

S_1S_0 = 11



(g) Decrement of *A*



(h) Transfer of *A*

Fig. 9.6

The condition illustrated in Fig. 9.6 (g) inserts all 1s into the B terminals. This produces the decrement operation, F = A – 1. To show that this condition is indeed a decrement operation, consider a parallel adder with *n* full-adder circuits. When the **output carry** is 1, it **represents the number 2^n** because 2^n in binary consists of a 1 followed by *n* 0s. Subtracting 1 from 2^n, we obtain $2^n – 1$, which in binary is a number of *n* 1s. Adding $2^n – 1$ to A, we obtain $F = A + 2^n – 1 = 2^n + A – 1$. If the output carry 2^n is removed then we obtain the decrement operation, F = A – 1.

To demonstrate with a numerical example, let us consider, n = 8 and A = 9. Then

$$A = 0000\ 1001 = 9$$

$$2^8 = 1\ 0000\ 0000 = 256$$

$$2^8 – 1 = 1111\ 1111 = 255$$

$$A + 2^8 – 1 = 1\ 0000\ 1000 = 256 + 8$$

Removing the output carry $2^n = 256$, we obtain 8 (i.e., 9 – 1). Thus, we have decremented A by 1 (from 9 to 8) by adding to it a binary number with all 1s.

Function Table for Arithmetic Circuit

Table 9.1 Function Table for the arithmetic circuit of Fig. 9.8

Function select			Y equals	Output equals	Function
s_1	s_0	C_{in}			
0	0	0	0	$F = A$	Transfer A
0	0	1	0	$F = A + 1$	Increment A
0	1	0	B	$F = A + B$	Add B to A
0	1	1	B	$F = A + B + 1$	Add B to A plus 1
1	0	0	$\overline{B}$	$F = A + \overline{B}$	Add 1's complement of B to A
1	0	1	$\overline{B}$	$F = A + \overline{B} + 1$	Add 2's complement of B to A
1	1	0	All 1's	$F = A - 1$	Decrement A
1	1	1	All 1's	$F = A$	Transfer A

Functions of Arithmetic Circuit

The circuit that controls input *B* to provide the functions illustrated in Fig. 9.6 is called a **true/complement, one/zero element**. The circuit is illustrated in Fig. 9.7.

Two selection lines s_1 and s_0 control the input of each *B* terminal. The diagram shows one typical input designated by Bi and an output designated by Y_i.

In a typical application, there are n such circuits for $i = 1, 2, 3, \ldots\ldots, n$.

As shown in the Table of Fig. 9.7, when both s_1 and s_0 are equal to zero, the output Y_i is equal to 0 regardless of the value of B_i.

When $s_1 = 0$ and $s_0 = 1$, the top AND gate generates the value of B_i while the bottom AND gate output is 0; thus, the output Y_i is equal to B_i.

With $s_1 = 1$ and $s_0 = 0$, the bottom AND gate generates the complemented value of B_i while the top AND gate output is 0; thus, the output Y_i is equal to B'_i.

Finally, when both $s_1 = 1$ and $s_0 = 1$, both the top and bottom AND gates generate the normal and complemented value of B_i respectively; thus, the output Y_i is equal to $B_i + B_i'$, which is equal to 1. These are illustrated in the Table of Fig. 9.7

True/Complement, One/Zero Circuit

- The values of the Y inputs to the full-adder circuits are a function of selection variables S_1 and S_0 .

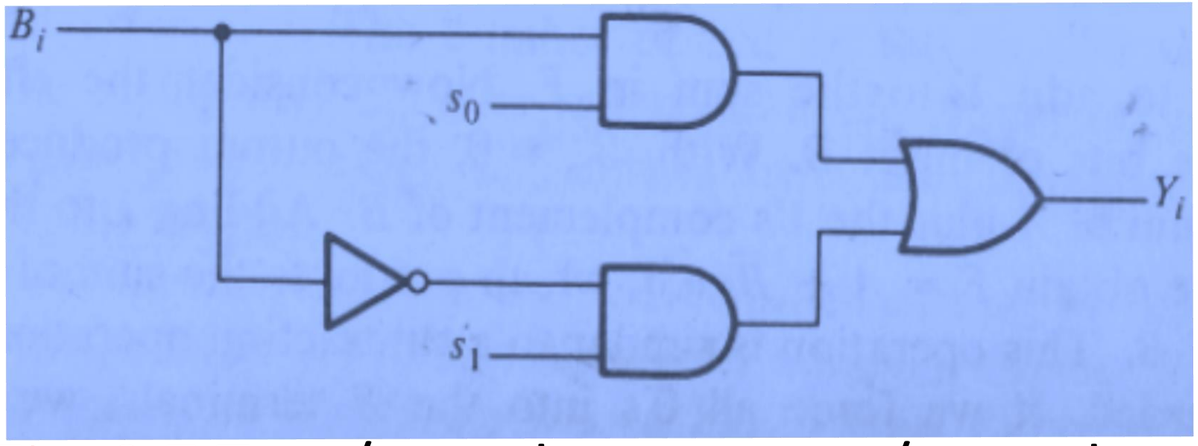


Fig. 9.7 True/complement, one/zero element

s_1	s_0	Y_i
0	0	0
0	1	B_i
1	0	B_i'
1	1	1

- The arithmetic circuit needs a **combinational circuit** in **each stage** specified by Boolean functions –

$$X_i = A_i$$

$$Y_i = B_i s_0 + B_i' s_1, \; i = 1,2,3,\ldots,n$$

Logic Diagram of an Arithmetic Circuit

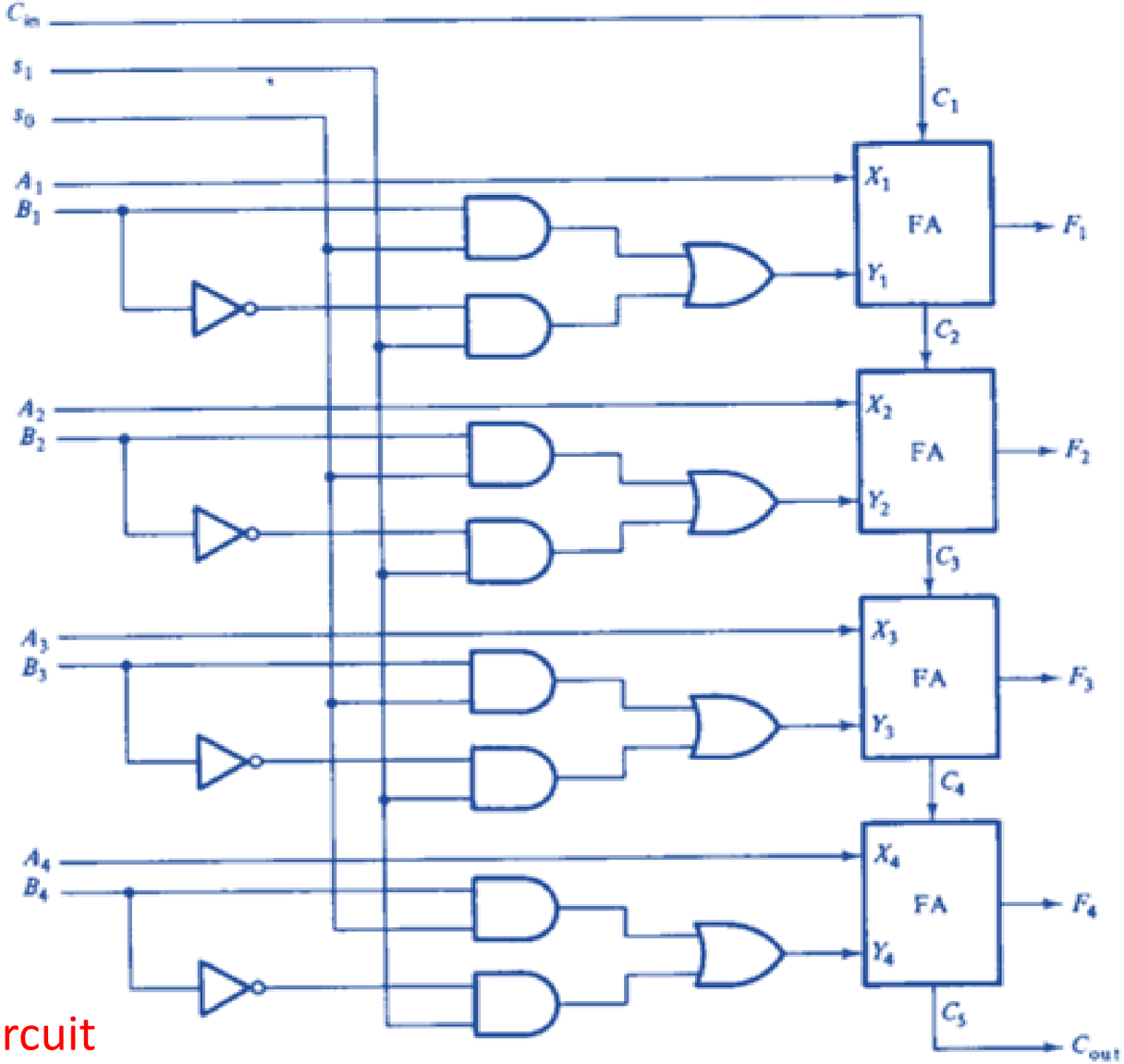


Figure 9-8 Logic Diagram of a 4-bit Arithmetic Circuit

Exercise (Homework)

- Design an adder/subtractor circuit with one selection variable s and two inputs A and B: when s = 0 the circuit performs A+B. When *s* = 1 the circuit performs A-B by taking the 2's complement of B.

$X_i = A_i$

$Y_i = B_i$ XOR s

$C_{in} = s$

Truth Table of XOR gate

INPUT		OUTPUT
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

s	*F*	*X*	*Y*	C_{in}
0	*A+B*	*A*	*B*	0
1	*A-B*	*A*	*B'*	1

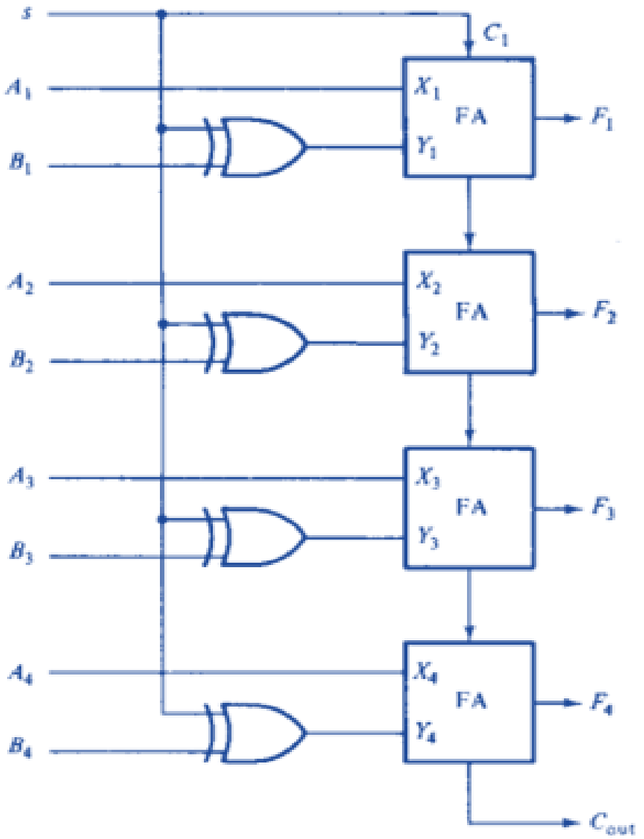
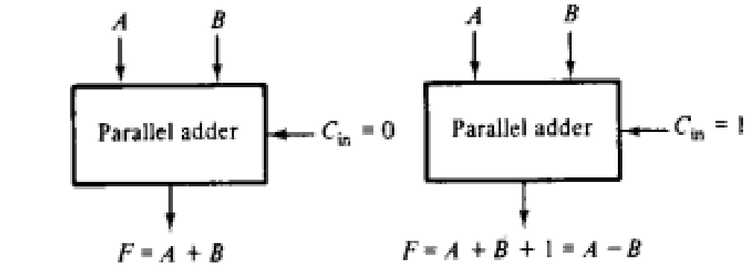
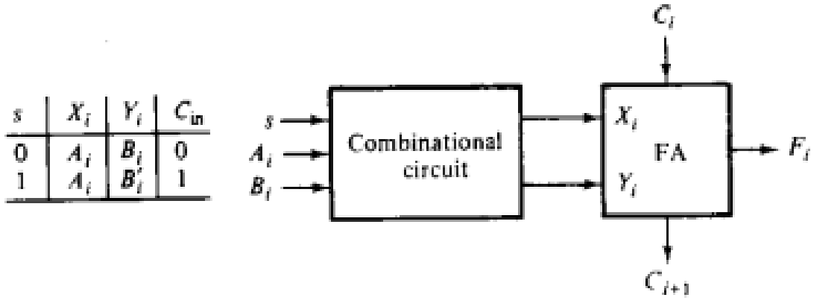


Figure 9-10 4-bit adder/subtractor circuit



(a) Function specification



(b) Specifying combinational circuit

s	A_i	B_i	X_i	Y_i
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

$X_i = A_i$

$Y_i = B_i \oplus s$

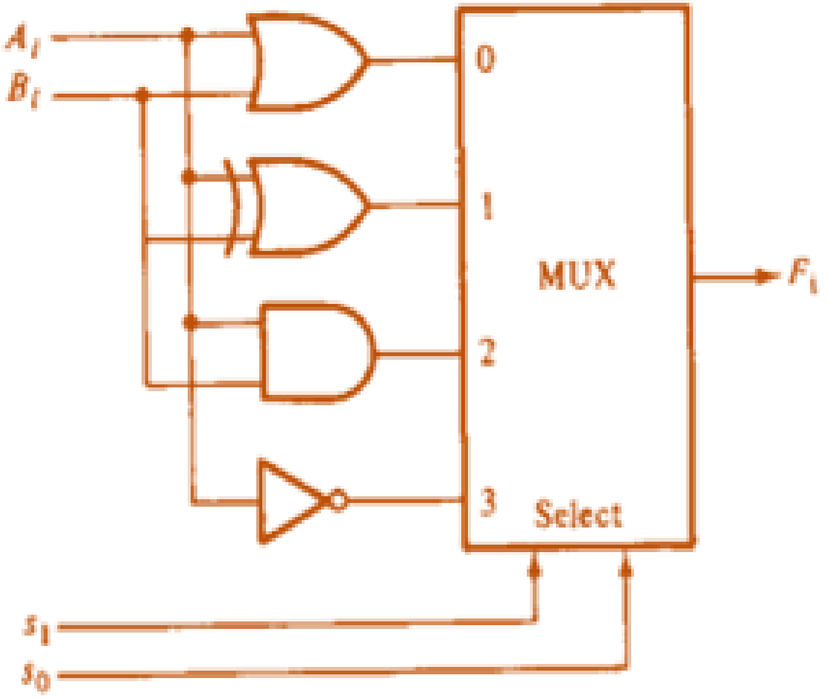
$C_{in} = s$

(c) Truth table and simplified equations

Figure 9-9 Derivation of an adder/subtractor circuit

Design of a Logic Circuit

- The logic microoperations manipulate the bits of the operands separately and treat each bit as a binary variable.
- All logic operations can be obtained by means of **AND, OR** and **NOT** operations.
- For 3 operations, we need 2 selection variables but 2 selection lines can choose between 4 logic operations and hence XOR operation was also incorporated in the logical circuit.



(a) Logic diagram

s_1	s_0	Output	Operation
0	0	$F_i = A_i + B_i$	OR
0	1	$F_i = A_i \oplus B_i$	XOR
1	0	$F_i = A_i B_i$	AND
1	1	$F_i = A_i'$	NOT

(b) Function table

Figure 9-11 One stage of logic circuit

Combining Logic and Arithmetic Circuits

- The logic circuit can be combined with the arithmetic circuit to produce one arithmetic logic unit.
- Selection variables s1 and s0 can be made common to both sections provided we use a third variable s2 to differentiate between the two.
- This third variable is the select input to a 2:1 multiplexer.

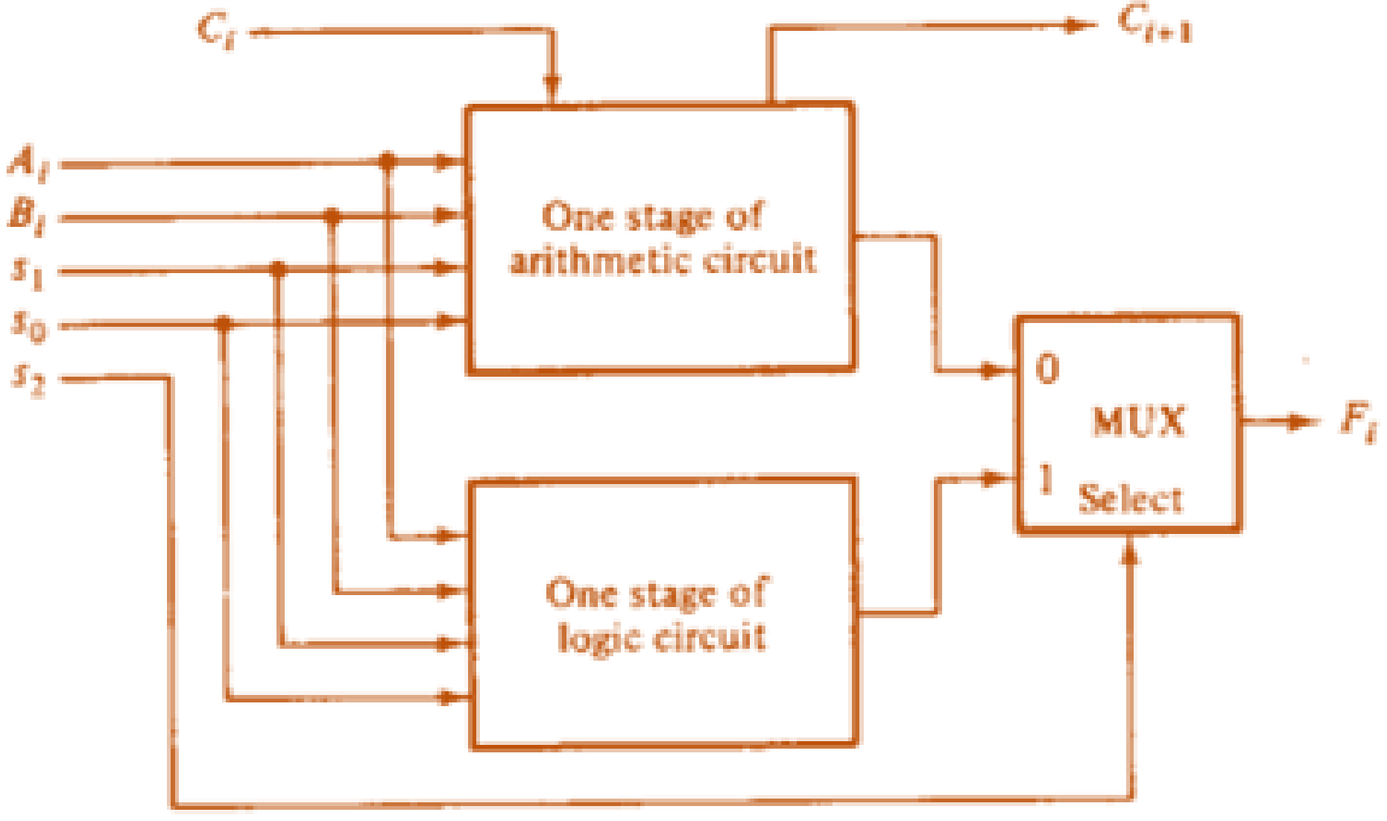


Figure 9-12 Combining logic and arithmetic circuits

Logic Operations in One Stage of Arithmetic Circuit

Table 9-3 Logic operations in one state of arithmetic circuit

s_2	s_1	s_0	X_i	Y_i	C_i	$F_i = X_i \oplus Y_i$	Operation		Required operation
1	0	0	A_i	0	0	$F_i = A_i$	Transfer A	?	OR
1	0	1	A_i	B_i	0	$F_i = A_i \oplus B_i$	XOR		XOR
1	1	0	A_i	B_i'	0	$F_i = A_i \odot B_i$	Equivalence	?	AND
1	1	1	A_i	1	0	$F_i = A_i'$	NOT		NOT

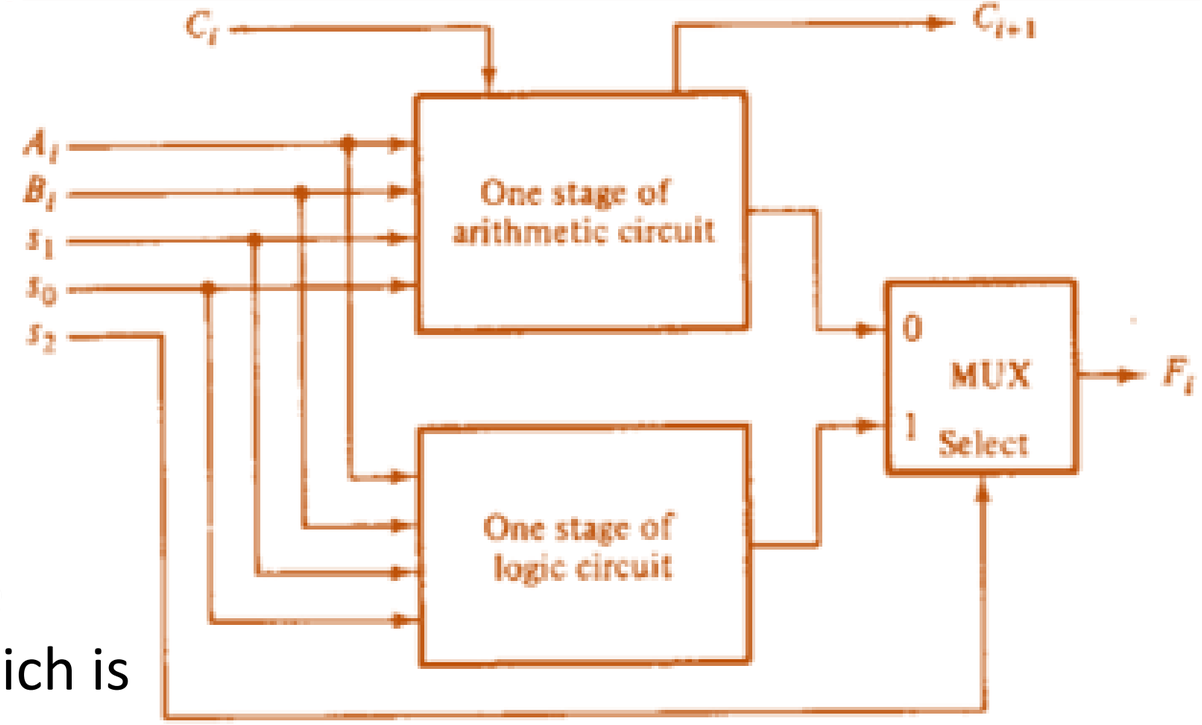


Figure 9-12 Combining logic and arithmetic circuits

XNOR gate gives an output "1" when its inputs are "*logically equal*" or "*equivalent*" to each other, which is why an **Exclusive-NOR** gate is sometimes called an **Equivalence Gate**.

Input		Output
A	***B***	***A* XNOR *B***
1	0	0
1	1	1

Problems of combining the Arithmetic and Logic Circuits in one stage

The value of X_i is always equal to input A_i. Table 9.3 shows the four logic operations obtained when a third selection variable $s_2 = 1$. This selection variable forces C_i to be equal to 0 while s_1 and s_0 choose a particular value for Y_i.
The four logic operations obtained by this configuration are transfer, exclusive OR (XOR), **equivalence (XNOR)**, and complement. The third entry is the **XNOR operation** because of the following-

$$A_i \oplus B_i' = A_i B_i + A_i' B_i' = A_i \odot B_i$$

The last entry in the table is the **NOT or complement operation** because of the following-

$$A_i \oplus 1 = A_i \text{AND} 0 + A_i' \text{AND} 1 = 0 + A_i' = A_i'$$

Design of Arithmetic Logic Unit

The design steps of an ALU are as follows:

1. Design an arithmetic section independent of the logic section
2. Determine the logic operations obtained from the arithmetic circuit in step 1, assuming the input carries of all stages are 0.
3. Modify the arithmetic circuit to obtain the required logic operation

Logic Diagram of an Arithmetic Circuit

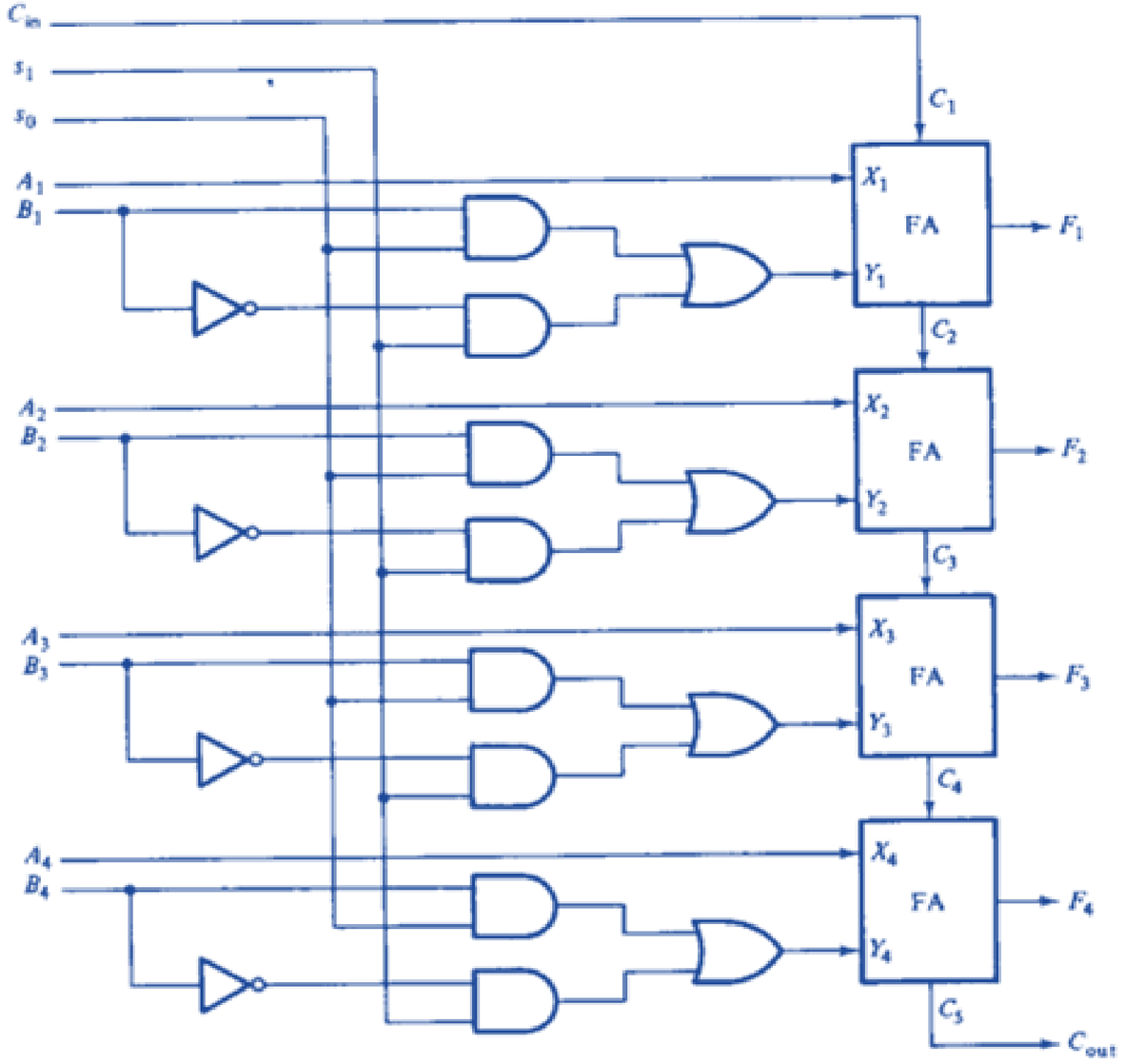


Figure 9-8 Logic Diagram of a 4-bit Arithmetic Circuit

Table 9-3

S_2	S_1	S_0	X_i	Y_i	C_i	$F_i = X_i \oplus Y_i$	Operation	Required operation
1	0	0	A_i	0	0	$F_i = A_i$	Transfer A	OR
1	0	1	A_i	B_i	0	$F_i = A_i \oplus B_i$	XOR	XOR
1	1	0	A_i	B_i'	0	$F_i = A_i \odot B_i$	Equivalence	AND
1	1	1	A_i	1	0	$F_i = A_i'$	NOT	NOT

Converting an Arithmetic Circuit to Logic Circuit

The solution to the first design step is shown in Fig. 9.8. The solution to the second design step is presented in Table 9.3. The solution of the third step is shown below. From Table 9.3, we see that when $s_2 = 1$, the input carry C_i in each stage must be 0. When both s_1 and s_0 are equal to 0, each stage generates the output function $F_i = A_i$. To change the output to an OR operation, we must change the input to each full-adder circuit from A_i to $A_i + B_i$. This can be accomplished by ORing A_i and B_i when $s_2s_1s_0 = 100$.

The other selection variable combinations that gives an **undesirable output** is **$s_2s_1s_0 =$ 110**. The present unit generates the output, $F_i = A_i \odot B_i$ as discussed in the previous slide. However, we want to generate the AND operation (i.e., $F_i = A_iB_i$) for this combination. For this purpose, let us investigate the possibility of ORing each input A_i with a Boolean variable K_i. The function so obtained is used for X_i when $s_2s_1s_0 = 110$.

Converting an Arithmetic Circuit to Logic Circuit

If the value of **K_i is ORed with input A_i** then we obtain (remember, when $s_2s_1s_0 = 110$ then $X_i = A_i$ and $Y_i = B_i'$),

$$F_i = X_i \oplus Y_i = (A_i + K_i) \oplus B_i' = (A_i + K_i)B_i'' + (A_i + K_i)'B_i'$$
$$= (A_i + K_i)B_i + A_i'K_i'B_i' = A_iB_i + K_iB_i + A_i'K_i'B_i'$$

Careful inspection of the result reveals that if the variable K_i is replaced by B_i' then the second and third terms will be eliminated from the final expression above, because $B_iB_i' = 0$ and thus we get the required AND operation between *A* and *B*.

$$F_i = X_i \oplus Y_i = (A_i + B_i') \oplus B_i' = (A_i + B_i')B_i'' + (A_i + B_i')'B_i'$$
$$= (A_i + B_i')B_i + A_i'B_i''B_i' = A_iB_i + B_i'B_i + A_i'B_i''B_i' = A_iB_i + 0 + 0 = A_iB_i$$

Therefore, we conclude that to get the required AND operation, we must make an OR operation between A_i and B_i' before applying it to the X_i input of the full-adder circuit when $s_2s_1s_0 = 110$ and Y_i input of the full-adder circuit will get B_i'.

Converting an Arithmetic Circuit to Logic Circuit

The final ALU is shown in Fig. 9.13. Only the first two stages are drawn, but the diagram can be easily extended to more stages. The inputs to each full-adder circuit are specified by the Boolean functions:

$$X_i = A_i + s_2 s_1' s_0' B_i + s_2 s_1 s_0' B_i'$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = s_2' C_i$$

When $\boldsymbol{s_2 = 0}$ then these three function reduce to-

$$X_i = A_i$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = C_i$$

These are the **functions of the arithmetic circuit** of Fig. 9.8

Converting an Arithmetic Circuit to Logic Circuit

When $s_2 = 1$ then these three function reduce to-

$$X_i = A_i + s_1' s_0' B_i + s_1 s_0' B_i'$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = 0$$

These are the **functions of the logic circuit** of Fig. 9.8.

When $s_2 s_1 s_0 = 100$, $X_i = A_i + s_1' s_0' B_i + s_1 s_0' B_i' = A_i + B_i + 0 = A_i + B_i$ and $Y_i = s_0 B_i + s_1 B_i' = 0 + 0 = 0$. Hence, we can get the OR operation at the output, $F_i = X_i \oplus Y_i = (A_i + B_i) \oplus 0 = A_i + B_i$.

When $s_2 s_1 s_0 = 101$, $X_i = A_i + s_1' s_0' B_i + s_1 s_0' B_i' = A_i + 0 + 0 = A_i$ and $Y_i = B_i + 0 = B_i$. Hence, we can get the required XOR operation at the output, $F_i = X_i \oplus Y_i = A_i \oplus B_i$.

Converting an Arithmetic Circuit to Logic Circuit

When $s_2 = 1$ then these three function reduce to-

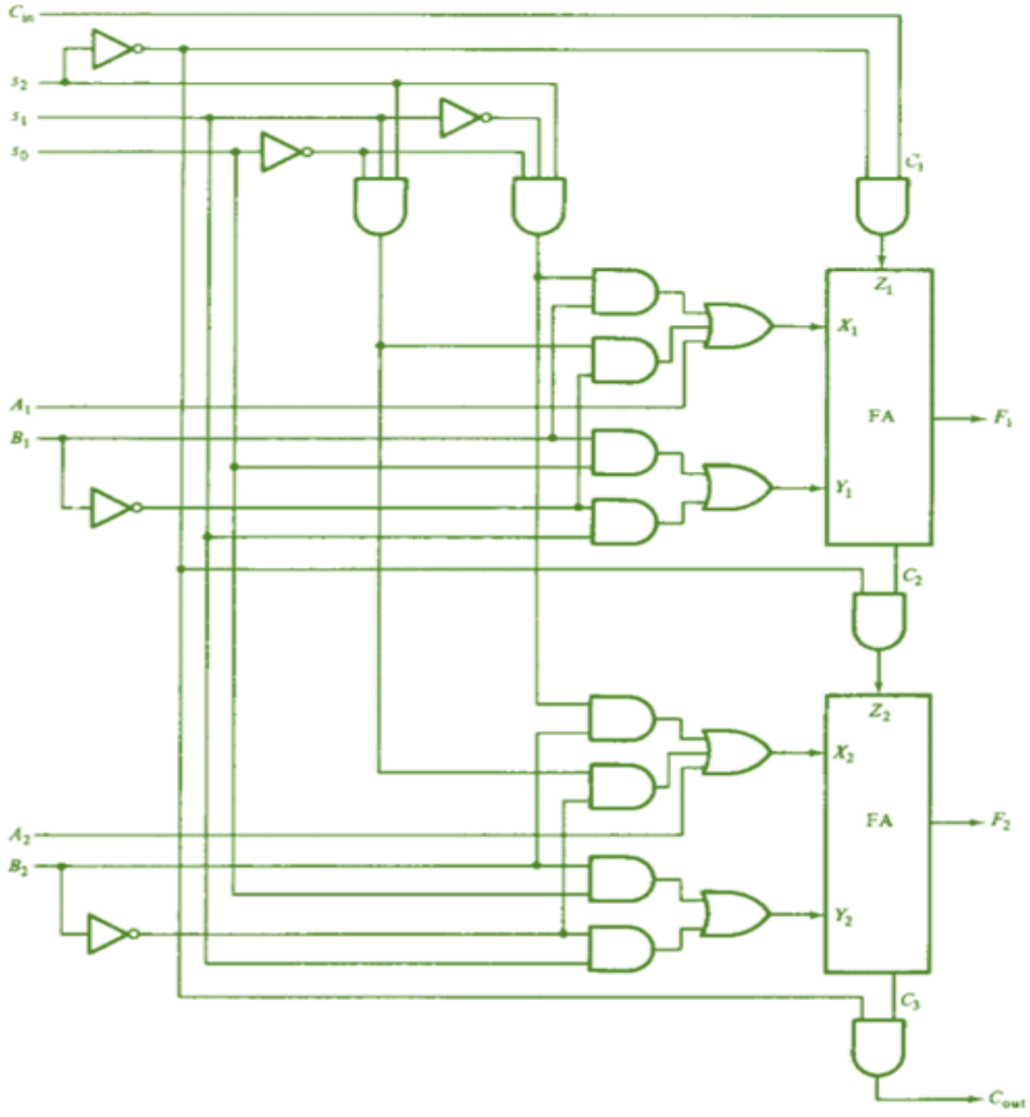
$$X_i = A_i + s_1' s_0' B_i + s_1 s_0' B_i'$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = 0$$

These are the **functions of the logic circuit** of Fig. 9.8.

When $s_2 s_1 s_0 = 110$, $X_i = A_i + s_1' s_0' B_i + s_1 s_0' B_i' = A_i + 0 + B_i' = A_i + B_i'$ and $Y_i = s_0 B_i + s_1 B_i' = 0 + B_i' = B_i'$. Hence, we can get the required AND operation at the output, $F_i = X_i \oplus Y_i = (A_i + B_i') \oplus B_i' = A_i B_i$.

When $s_2 s_1 s_0 = 111$, $X_i = A_i + s_1' s_0' B_i + s_1 s_0' B_i' = A_i + 0 + 0 = A_i$ and $Y_i = B_i + B_i' = 1$. Hence, we can get the required NOT operation at the output, $F_i = X_i \oplus Y_i = A_i \oplus 1 = A_i'$.

Converting an Arithmetic Circuit to Logic Circuit



Logic Diagram of ALU (Only the **first two stages** are drawn)

Table 9-3

S_2	S_1	S_0	X_i	Y_i	C_i	$F_i = X_i \oplus Y_i$	Operation	Required operation
1	0	0	A_i	0	0	$F_i = A_i$	Transfer A ?	OR
1	0	1	A_i	B_i	0	$F_i = A_i \oplus B_i$	XOR	XOR
1	1	0	A_i	B_i'	0	$F_i = A_i \odot B_i$	Equivalence ?	AND
1	1	1	A_i	1	0	$F_i = A_i'$	NOT	NOT

Figure 9.13: Logic diagram of ALU

Converting an Arithmetic Circuit to Logic Circuit

The 12 operations (8 arithmetic and 4 logical) generated in the ALU are summarized in Table 9.4. The function is selected through s_2, s_1, s_0, and C_{in}.

The arithmetic operations are identical to the ones listed for the arithmetic circuit. The value of C_{in} for the four logic functions has no effect on the operations of the unit and those entries are marked as don't cares (*X*s).

Table 9.4

Selection					
s_2	s_1	s_0	C_{in}	Output	Function
0	0	0	0	$F = A$	Transfer A
0	0	0	1	$F = A + 1$	Increment A
0	0	1	0	$F = A + B$	Addition
0	0	1	1	$F = A + B + 1$	Add with carry
0	1	0	0	$F = A - B - 1$	Subtract with borrow
0	1	0	1	$F = A - B$	Subtraction
0	1	1	0	$F = A - 1$	Decrement A
0	1	1	1	$F = A$	Transfer A
1	0	0	X	$F = A \vee B$	OR
1	0	1	X	$F = A \oplus B$	XOR
1	1	0	X	$F = A \wedge B$	AND
1	1	1	X	$F = \bar{A}$	Complement A

Design of Arithmetic Logic Circuit

[Supplement Starts]

Design a 2-bit ALU for the operations listed below in the Table:

Binary Code			Function of selection variables			
	B	A	F with C_{in} = 0	F with C_{in} = 1	D	H
0 0 0	Input Data	Input Data	A	A+1	None	Circulate-Left with Carry
0 0 1	R1	R1	A+B	A+B+1	R1	Circulate-Right with Carry
0 1 0	R2	R2	A+B`	A+B`+1	R2	No shift
0 1 1	R3	R3	A-1	A	R3	0's to the output Bus
1 0 0	R4	R4	A OR B	A OR B	R4	-
1 0 1	R5	R5	A XOR B	A XOR B	R5	Shift Left with I_L=0
1 1 0	R6	R6	A AND B	A AND B	R6	Shift Right with I_R=0
1 1 1	R7	R7	A'	A'	R7	1's to the output Bus

Design of Arithmetic Logic Circuit

TABLE 9-4 Function table for the ALU of Fig. 9-13

Selection				Output	Function
s_2	s_1	s_0	C_{in}		
0	0	0	0	$F = A$	Transfer A
0	0	0	1	$F = A + 1$	Increment A
0	0	1	0	$F = A + B$	Addition
0	0	1	1	$F = A + B + 1$	Add with carry
0	1	0	0	$F = A - B - 1$	Subtract with borrow
0	1	0	1	$F = A - B$	Subtraction
0	1	1	0	$F = A - 1$	Decrement A
0	1	1	1	$F = A$	Transfer A
1	0	0	X	$F = A \vee B$	OR
1	0	1	X	$F = A \oplus B$	XOR
1	1	0	X	$F = A \wedge B$	AND
1	1	1	X	$F = \overline{A}$	Complement A

X_i	Y_i	Z_i
A	0	0
A	0	1
A	B	0
A	B	1
A	B’	0
A	B’	1
A	1	0
A	1	1
A+B	0	0
A	B	0
A+B’	B’	0
A	1	0

Design of Arithmetic Logic Circuit (K-maps)

For an arithmetic circuit, we don't need to change *A* for *X*, but for a logic circuit, we need to change *A* and we don't need C_{in}. So, we need to use only three variables (s_2, s_1, and s_0) in the K-map.

S_2S_1 \ S_0	0	1
00	*A*	*A*
01	*A*	*A*
11	*A*+*B'*	*A*
10	*A*+*B*	*A*

$$X_i = A_i + s_2 s_1' s_0' B_i + s_2 s_1 s_0' B_i'$$

For both arithmetic and logic circuits, we need to change *B* for *Y* for the same combination of s_2, s_1, and s_0, but for a logic circuit, we don't need C_{in}. So, we need to use only 3 variables (s_2, s_1, and s_0) in the K-map.

S_2S_1 \ S_0	0	1
00	0	B
01	B'	1
11	B'	1
10	0	B

$$Y_i = s_0 B_i + s_1 B_i'$$

For the arithmetic circuit, we need C_{in} for all combinations of s_2, s_1, and s_0, but for a logic circuit, we don't need C_{in}. So, we need to use all 4 variables (s_2, s_1, s_0, and C_{in}) in the K-map to force C_{in} to zero for *Z* for the same combination of s_2, s_1, and s_0.

S_2S_1 \ S_0C_{in}	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	0	0
10	0	0	0	0

$$Z_i = s_2' C_{in}$$

Design of Arithmetic Logic Circuit

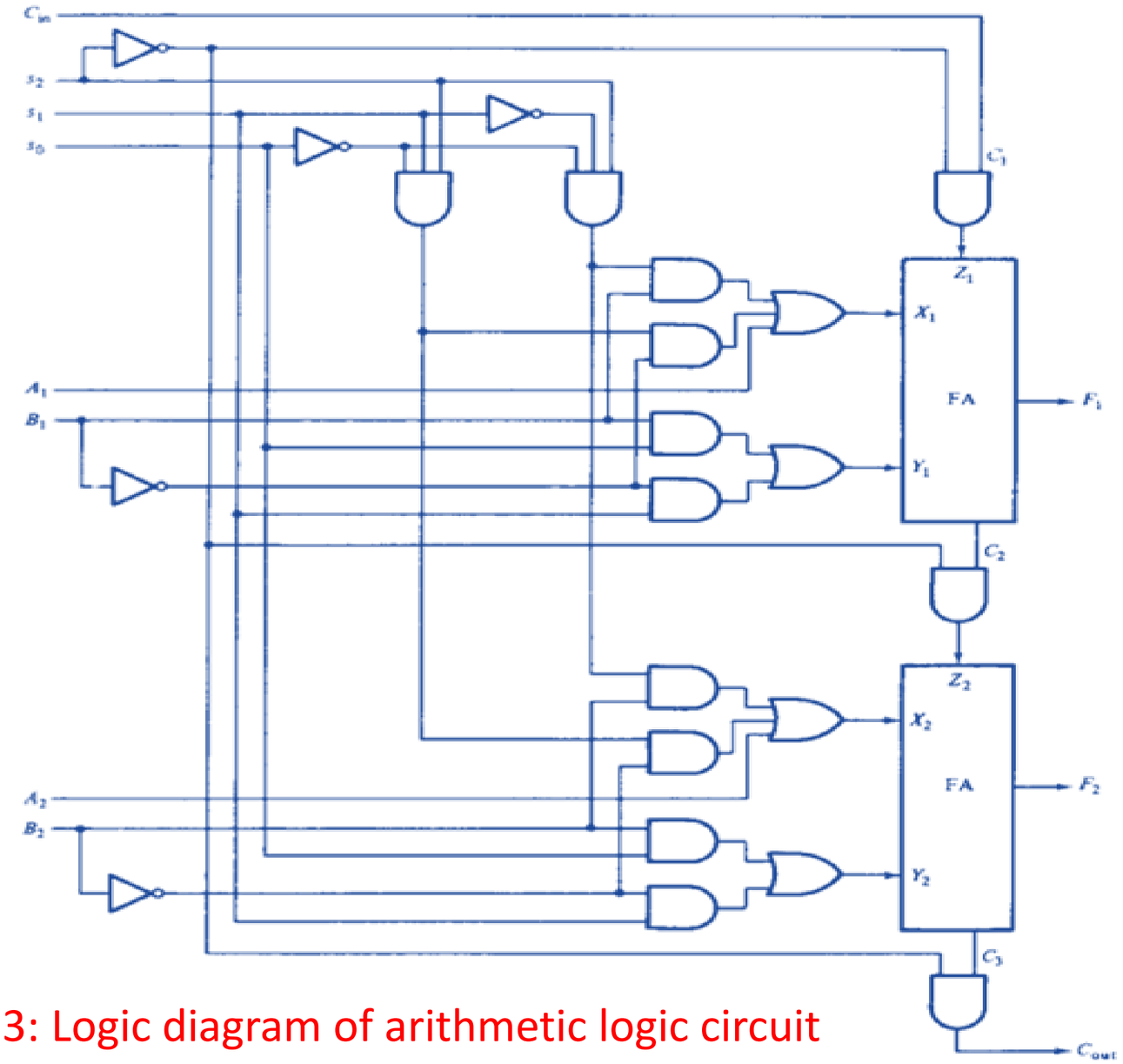


Figure 9.13: Logic diagram of arithmetic logic circuit

Next Class…..

- Status Register
- Shifter
- Design of a processor unit with control variables
- Micro operations for processor

Thanks for attending….