Computer Architecture
Execution Unit

Moniruzzaman

Adjunct Lecturer
North Western University, Khulna-9000

Confession

- Most of the materials have been collected from Internet.
- Images are taken from Internet.
- Various books are used to make these slides.
- Various slides are also used.
- References & credit:
  - Atanu Shome, Assistant Professor, CSE, KU.
  - Computer Organization and Design: the Hardware/Software Interface - Textbook by David A Patterson and John L. Hennessy.
  - Computer Organization and Architecture - Book by William Stallings

moniruzzaman

Execution Unit

An execution unit (EU) is a component of a computer's hardware that performs instructions, also known as functional units. EUs are separate blocks that perform arithmetic, logic, branching, and other actions. They work in parallel and can work on a different task at the same time as other parts.

Multiplexer

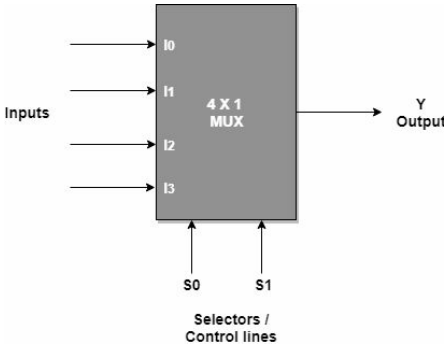
A multiplexer is a combinational circuit that has **many data inputs and a single output**, depending on control or select inputs.

For N input lines, log2(N) selection lines are required,
or equivalently, for 2^n input lines, n selection lines are needed.

Multiplexers are also known as "**N-to-1 selectors**," parallel-to-serial converters, many-to-one circuits, and universal logic circuits.

They are mainly used to increase the amount of data that can be sent over a network within a certain amount of time and bandwidth.

Multiplexer

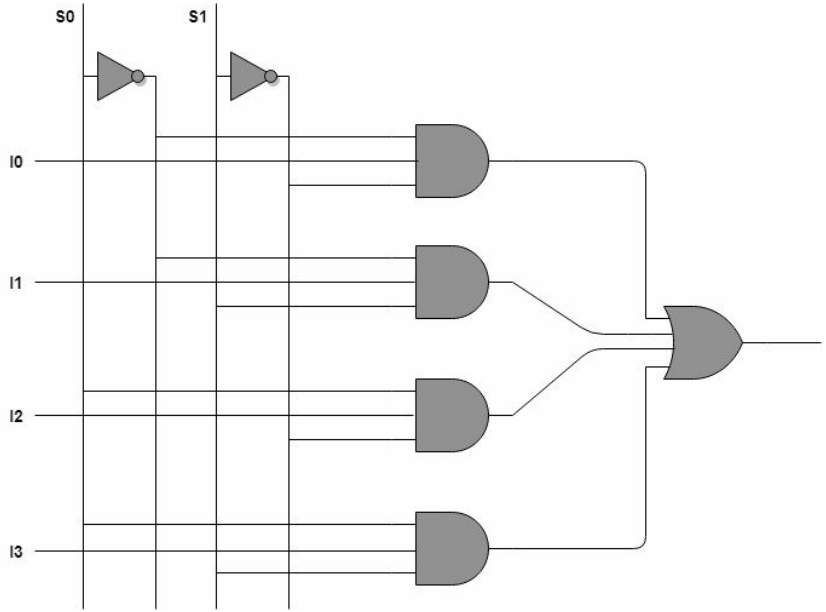


Truth Table

S0	S1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

So, final equation,

$Y = S0'.S1'.I0 + S0'.S1.I1 + S0.S1'.I2 + S0.S1.I3$



Registers

General-purpose registers (GPRs) and dedicated registers are two types of registers in a CPU.

GPRs are special memory cells that can be accessed quickly and temporarily store data and control information.

They are not dedicated to specific tasks and can be used for a wide range of instructions.

GPRs can store both data and addresses, and in some architectures, they can also store floating-point numbers.

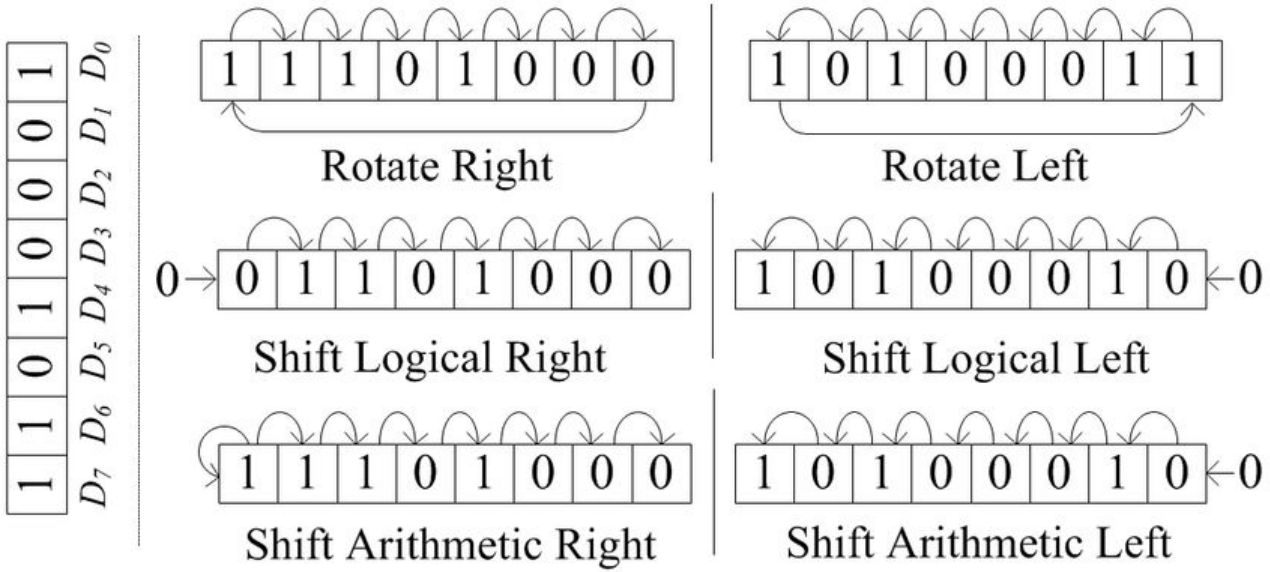
Registers

Dedicated registers are reserved for a specific purpose or role essential to the running of the processor.

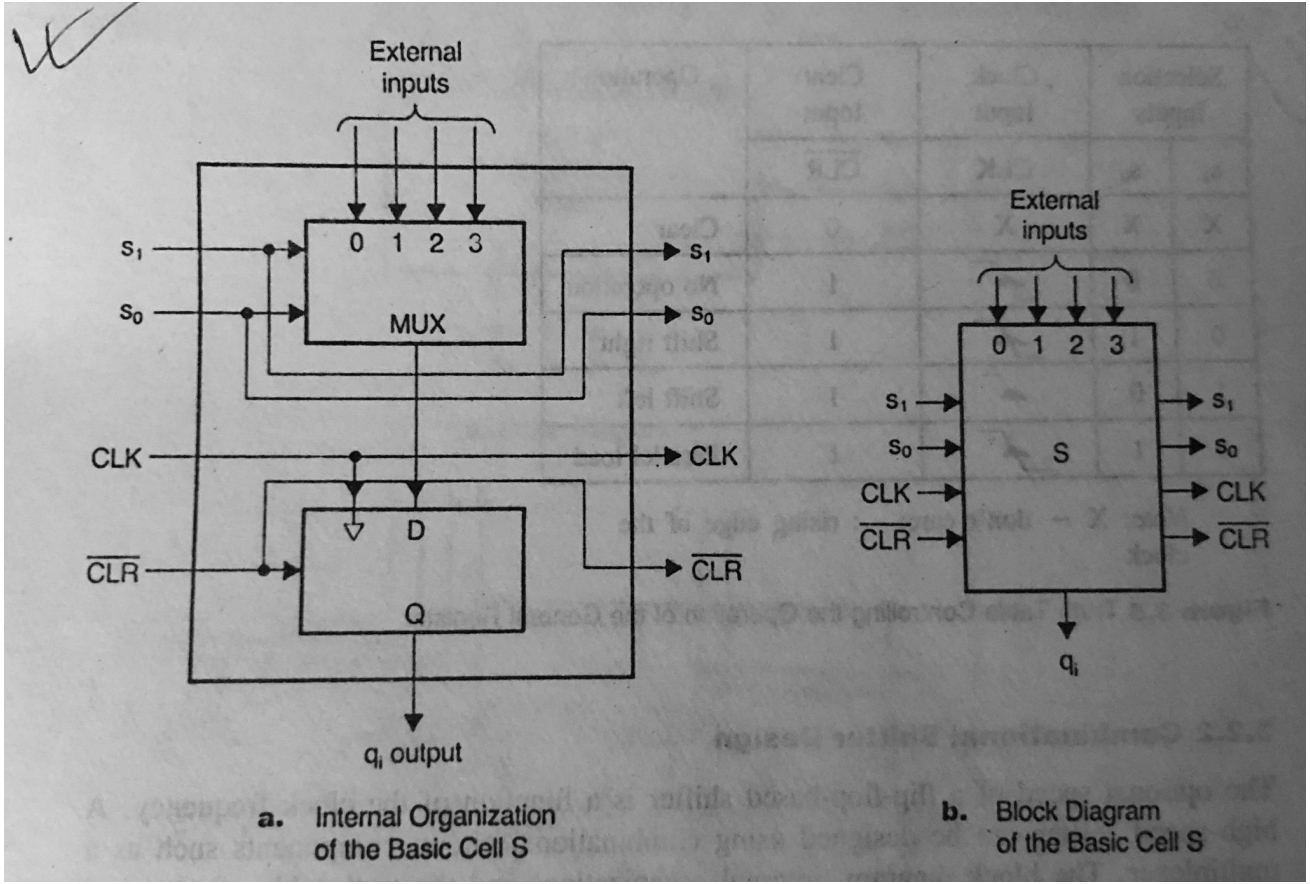
Here are some examples of special-purpose registers:

- Program Counter (PC): Keeps track of the next instruction to be executed
- Instruction Register (IR): Holds the current instruction being executed
- Stack pointer (SP): Keeps track of program flow and manages the stack
- Status register (SR/FLAGS): Stores flags indicating the status of operations

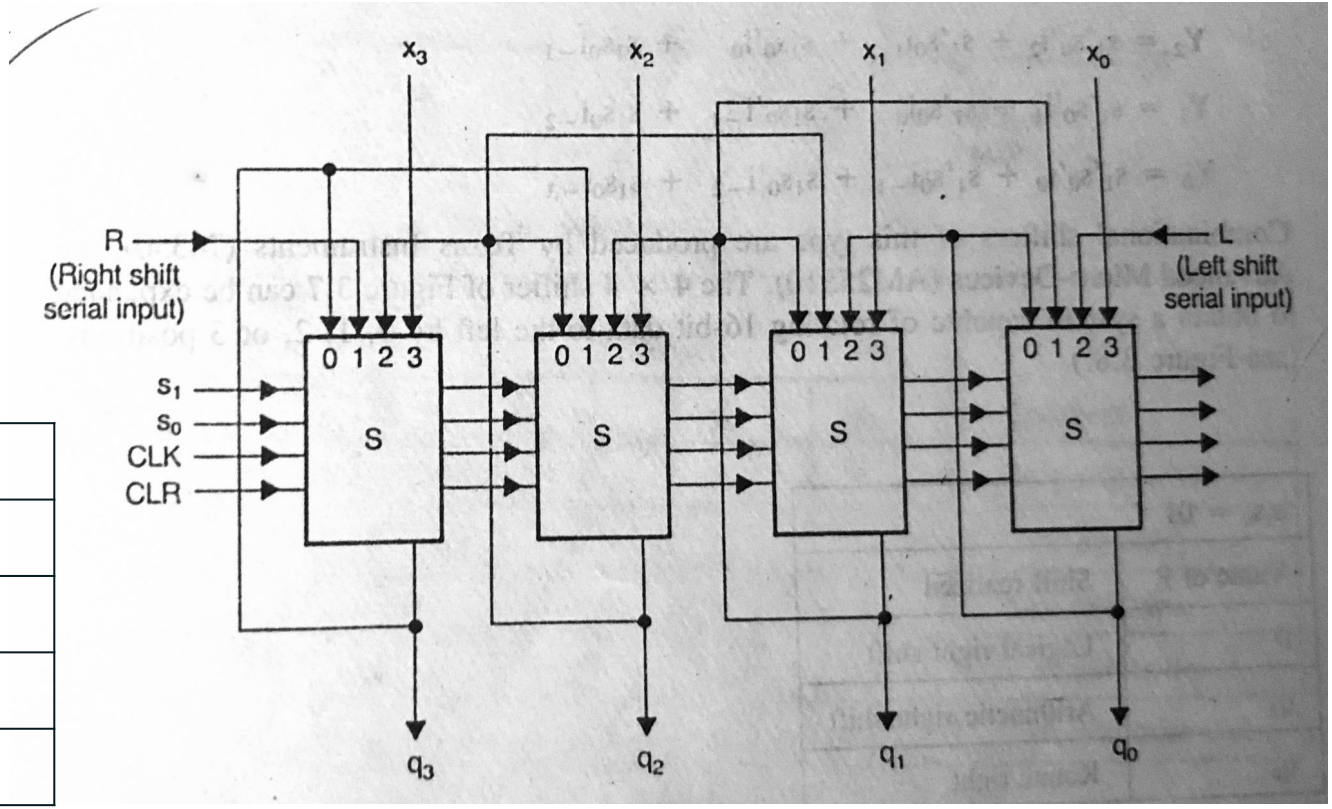
Shift Operation



General purpose registers



4 BIT General Purpose Register



s1	s0	Operation
0	0	No operation
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Adder

One-bit full adder

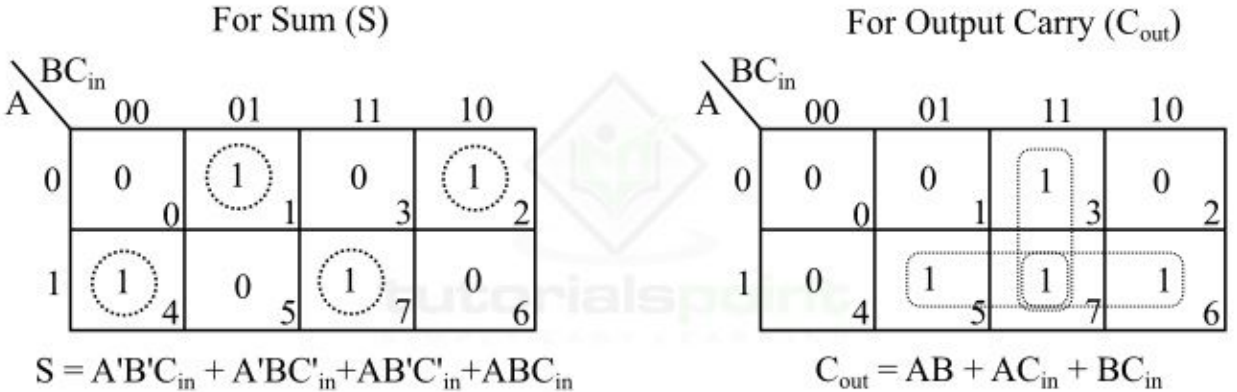


Figure 2 - K Map for Full Adder

Table 1: Full adder truth table.

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Sum, S = $A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in} = A \oplus B \oplus C_{in}$

Carry, C = $AB + AC_{in} + BC_{in}$

One-bit full adder Or Ripple Carry Adder

$$\text{Sum, } S = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in} = A \oplus B \oplus C_{in}$$

$$\text{Carry, } C = AB + AC_{in} + BC_{in}$$

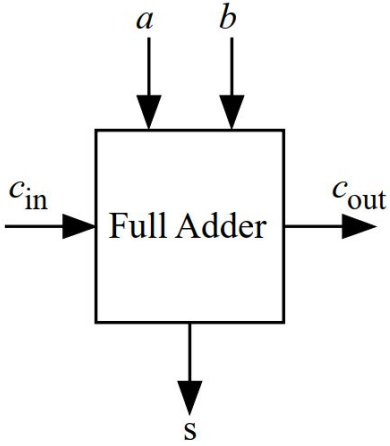


Figure 1: One-bit full adder.

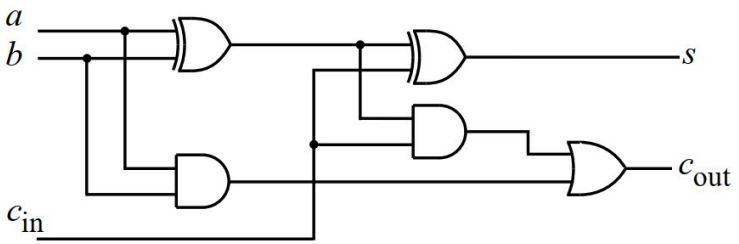
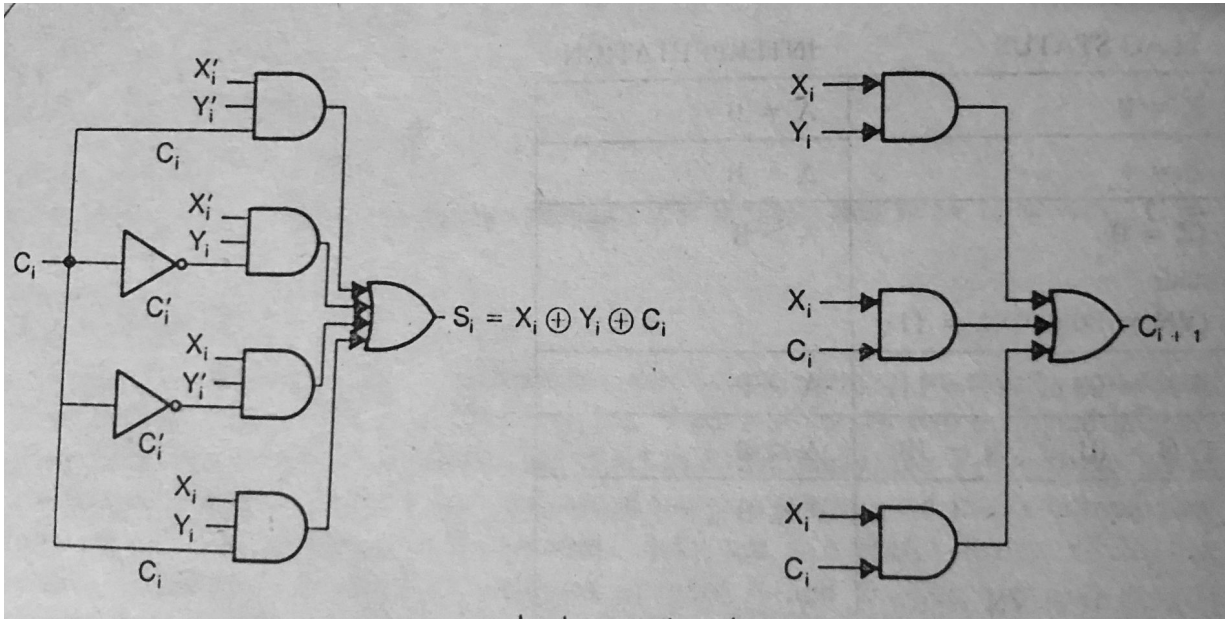


Figure 2: Gate implementation of full adder.

One-bit full adder Or Ripple Carry Adder



4-BIT Ripple Carry Adder

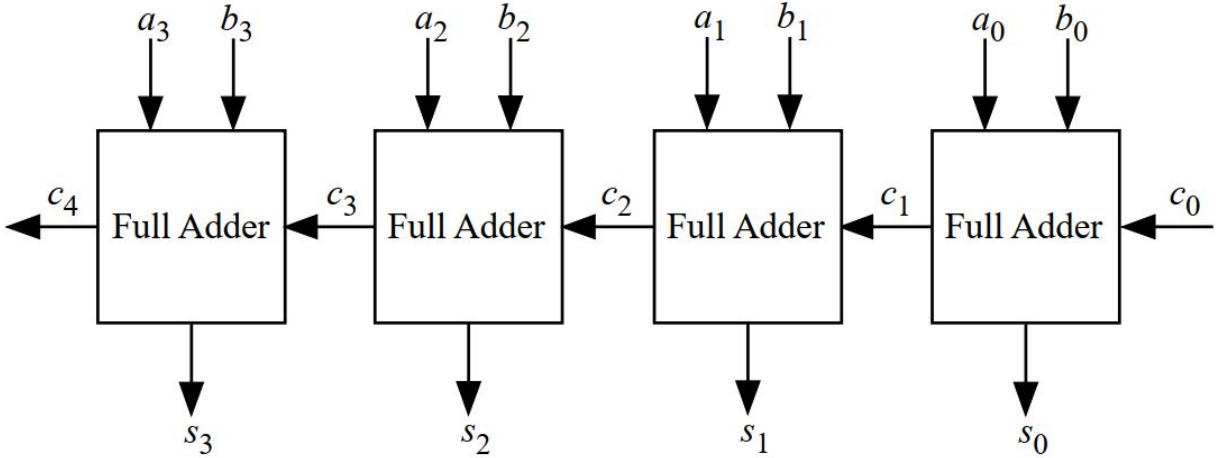
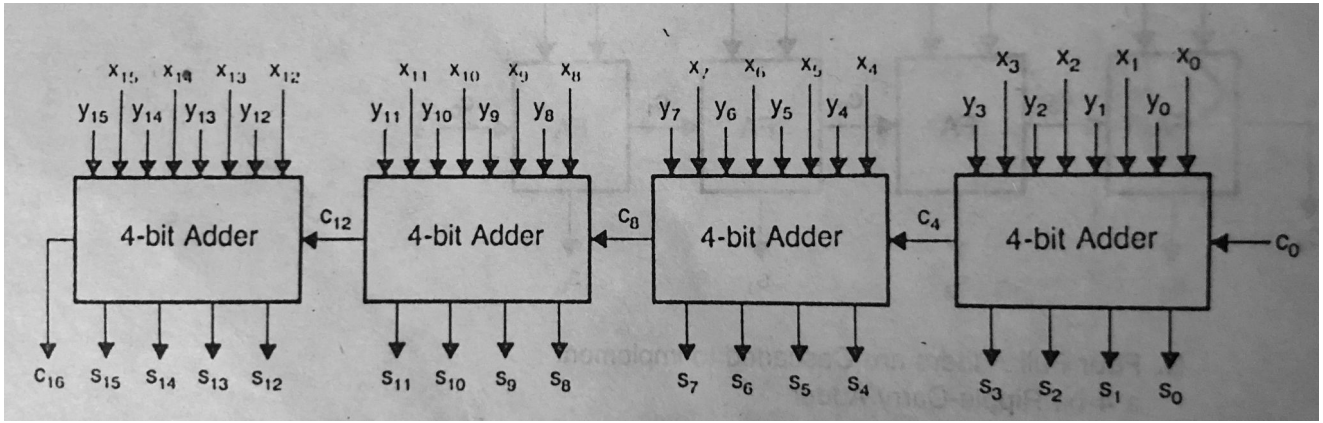


Figure 3: 4-bit full adder.

16-BIT Ripple Carry Adder



Carry Look-Ahead Adder

Carry Look Ahead Adder

$$c_{i+1} = a_i.b_i + (a_i \oplus b_i).c_i \quad (3)$$

$$s_i = (a_i \oplus b_i) \oplus c_i \quad (4)$$

The above two equations can be written in terms of two new signals P_i and G_i, which are shown in Figure 4:

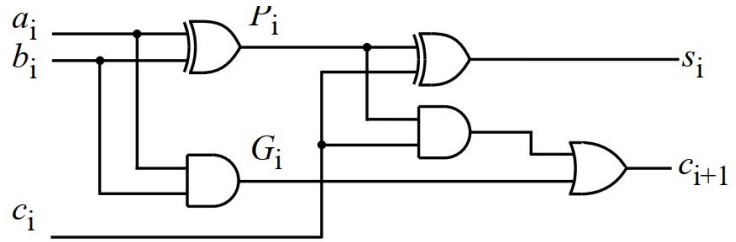


Figure 4: Full adder at stage i with P_i and G_i shown.

$$c_{i+1} = G_i + P_i.c_i \quad (5)$$

$$s_i = P_i \oplus c_i \quad (6)$$

where

$$G_i = a_i.b_i \quad (7)$$

$$P_i = a_i \oplus b_i \quad (8)$$

Carry Look Ahead Adder

G_i and P_i are called the carry generate and carry propagate terms, respectively. Notice that the generate and propagate terms only depend on the input bits and thus will be valid after one and two gate delay, respectively. If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple through all the previous stages to find its proper value.

Let's i = 0, 1, 2, 3 apply this to a -bit adder to make it clear. Putting in Equation 5, we get

$$c_1 = G_0 + P_0.c_0 \quad (10)$$

$$c_2 = G_1 + P_1.G_0 + P_1.P_0.c_0 \quad (11)$$

$$c_3 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.c_0 \quad (12)$$

$$c_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.c_0 \quad (13)$$

Carry Look Ahead Adder

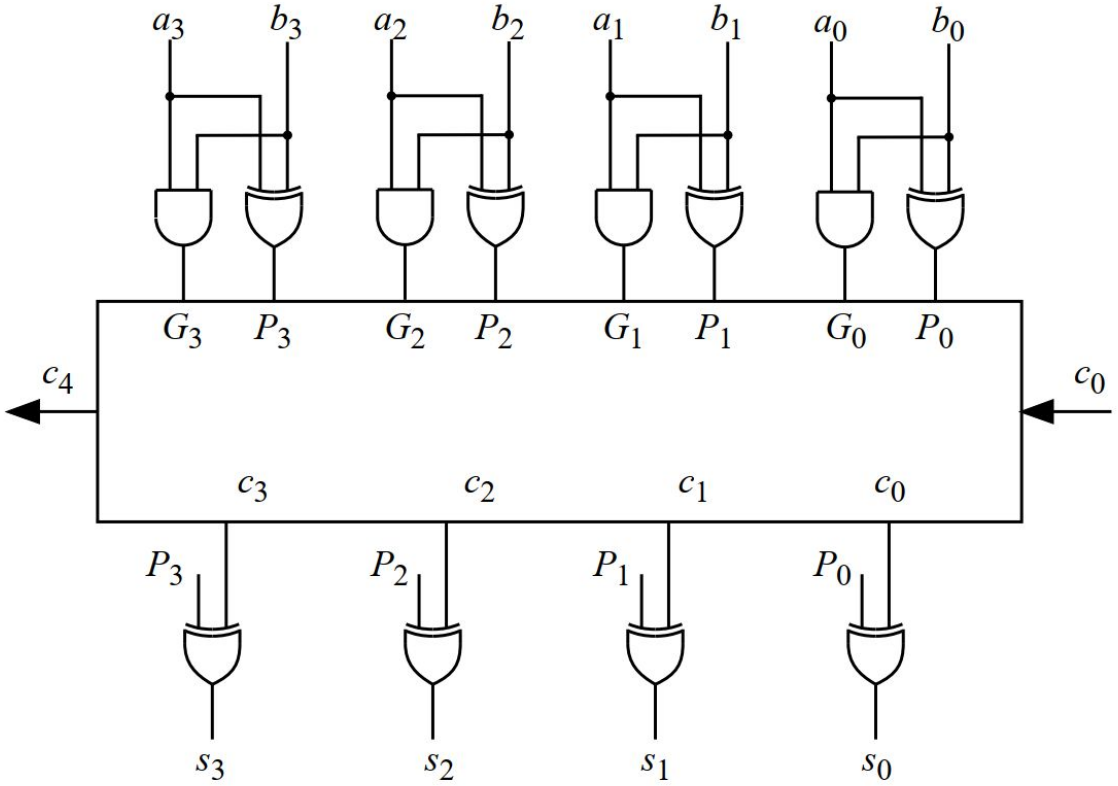
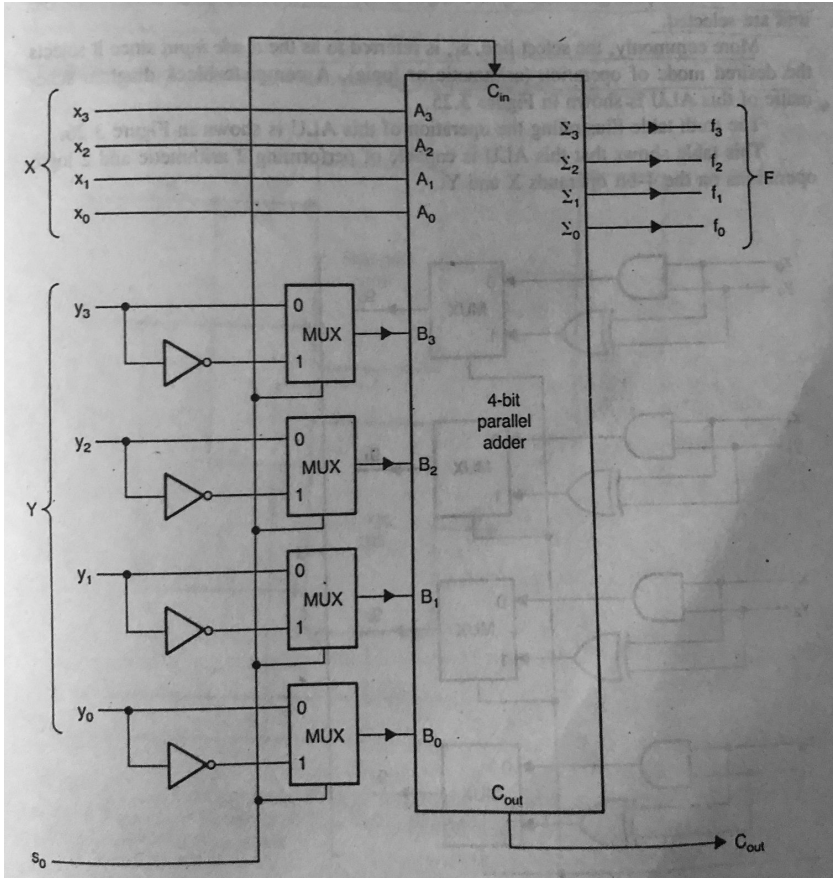


Figure 5: 4-Bit carry lookahead adder implementation detail.

ALU Design

4 BIT Arithmetic Unit



4 BIT Logic Unit

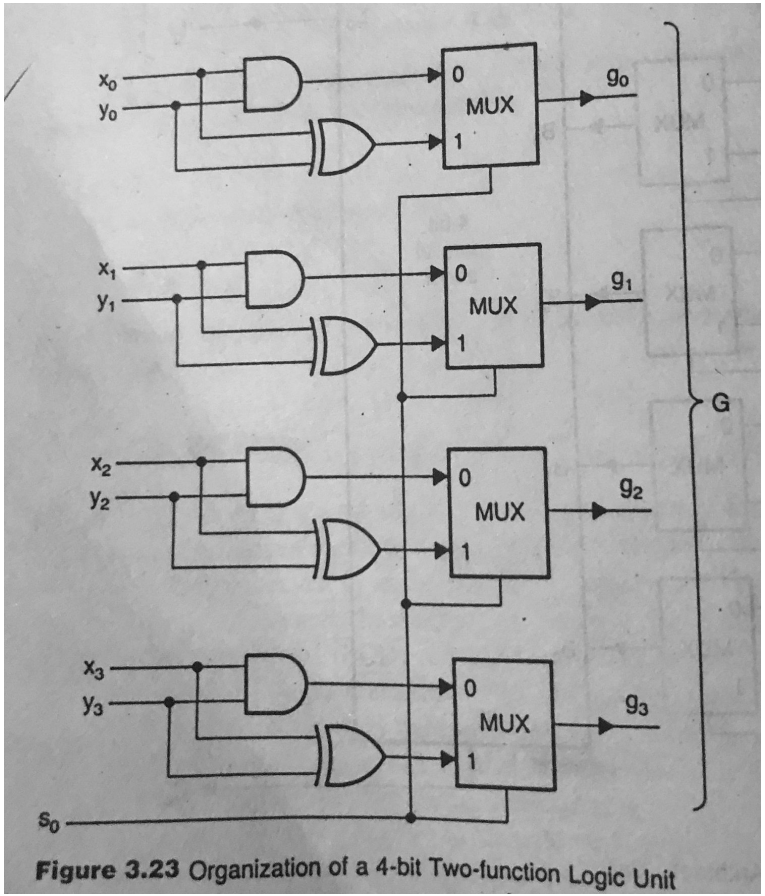
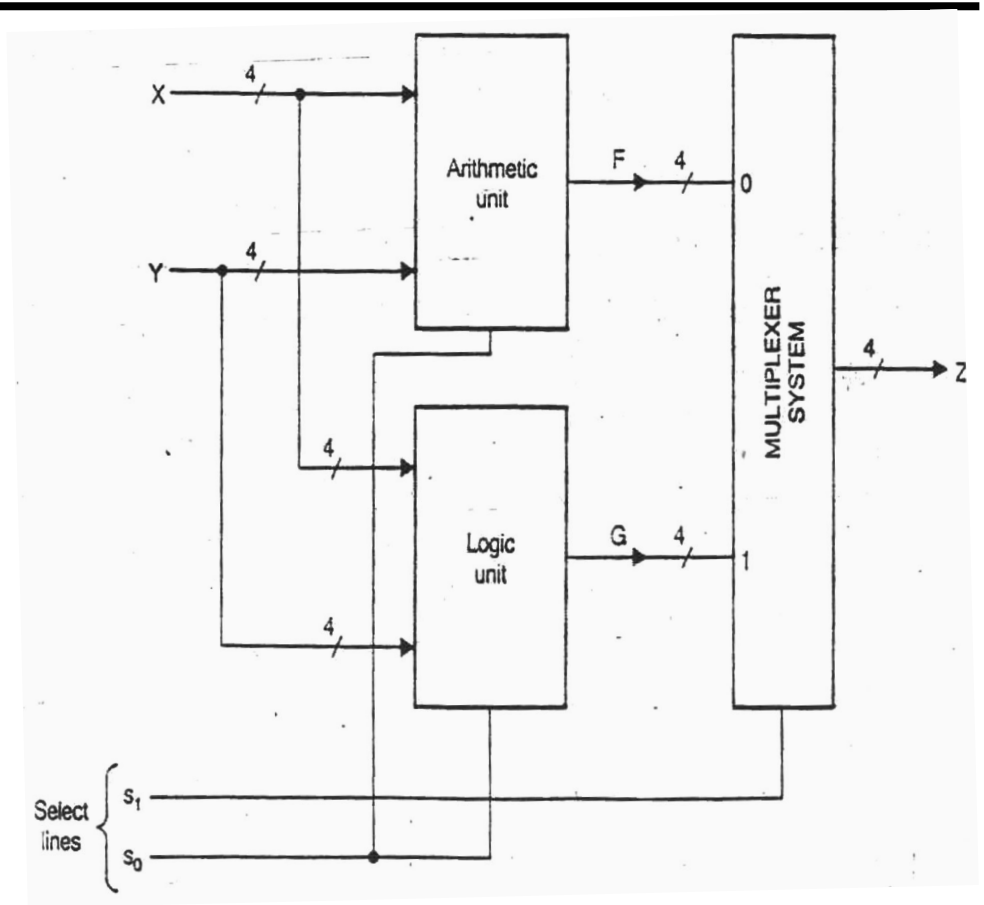


Figure 3.23 Organization of a 4-bit Two-function Logic Unit

4 BIT Logic Unit

Select lines		Output
s1	s0	
0	0	X + Y
0	1	X +Y+1
1	0	X And Y
1	1	X Ex-OR Y



Thank You