b) Arithmetic micro-operation with overflow detection.

1. Unsigned Binary Addition overflow

When the "Binary Addition Algorithm" is used with **unsigned** binary integer representation: The result is CORRECT only if the **CARRY OUT** of the high order column is **ZERO**. Unsigned overflow occurred when carry out =1; For example:

```
1
 1101 0010    210₁₀
 0110 1101    109₁₀
 ---------    -----
 0011 1111     63₁₀      The carry bit of 1 indicates overflow.
```

2. Signed Binary Addition overflow

There are many schemes for representing negative integers with patterns of bits. Two's complement is one of many ways to represent negative integers with bit patterns. With **two's complement** representation the result of addition is correct if the ***carry into*** the ***high order column*** is the ***same as the carry out of the high order column***. Overflow is detected by comparing these two bits. Here are some more examples:

No Overflow	No Overflow	Overflow	Overflow
11111 111 0011 1111 (63_{10}) 1101 0101 (-43_{10}) 0001 0100 (20_{10})	00000 011 1100 0001 (-63_{10}) 0010 1011 (43_{10}) 1110 1100 (-20_{10})	01111 100 0011 1111 (63_{10}) 0110 0100 (100_{10}) 1010 0011 (-93_{10})	10000 000 1100 0001 (-63_{10}) 1001 1100 (-100_{10}) 0101 1101 (93_{10})

The truth table of tow's sign bits (A_{sign} and B_{sign} bit) is shown below:

INPUTS			**OUTPUTS**		
$\mathbf{A_{sign}}$	$\mathbf{B_{sign}}$	**CARRY IN**	**CARRY OUT**	$\mathbf{SUM_{sign}}$	**OVERFLOW**
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

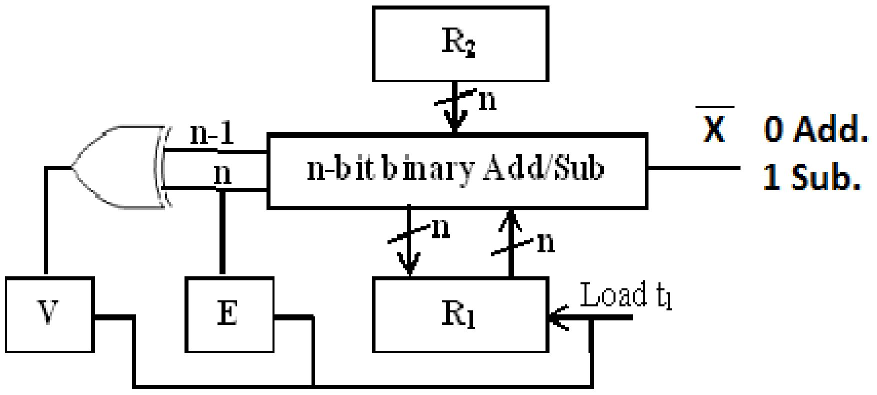
From the above truth table:

- Notice that **overflow** occurs only when: $\mathbf{CARRY_{in} \neq CARRY_{out}}$
- or simply: $\mathbf{V = C_{in}}$ **XOR** $\mathbf{C_{out}}$; where **V** is the overflow signal.

So that, the Arithmetic micro-operation with overflow detection can be design as:

t_1X: $ER_1 \leftarrow R_1+R_2$

t_1X: $ER_1 \leftarrow R_1+R_2+1$



Each of the **arithmetic micro operations** can be implemented in one **composite** arithmetic circuit. This circuit comprised of:

- Parallel full adders and
- Multiplexers are used to choose between the different operations.

The multiplexer controls which data is fed into input of the adder (suppose B, A represent that inputs). The output of the binary adder is computed from

$$D = A + B + C_{in}$$

The B input can have one of 4 different values: B,$\overline{B}$, always "1", or always "0"

Figure (7); below represent 4-bit arithmetic circuit.

S_1	S_2	C_{in}	Input B	$D = A+B+C_{in}$	Operation
0	0	0	B	A+B	ADD
0	0	1	B	A+B+1	ADD with carry
0	1	0	$\overline{B}$	A+$\overline{B}$	Sub with borrow (1's comp)
0	1	1	$\overline{B}$	A+$\overline{B}$+1	Sub in 2's comp.
1	0	0	0	A	Transfer
1	0	1	0	A+1	Increment
1	1	0	1	A-1	Decrement
1	1	1	1	A	Transfer

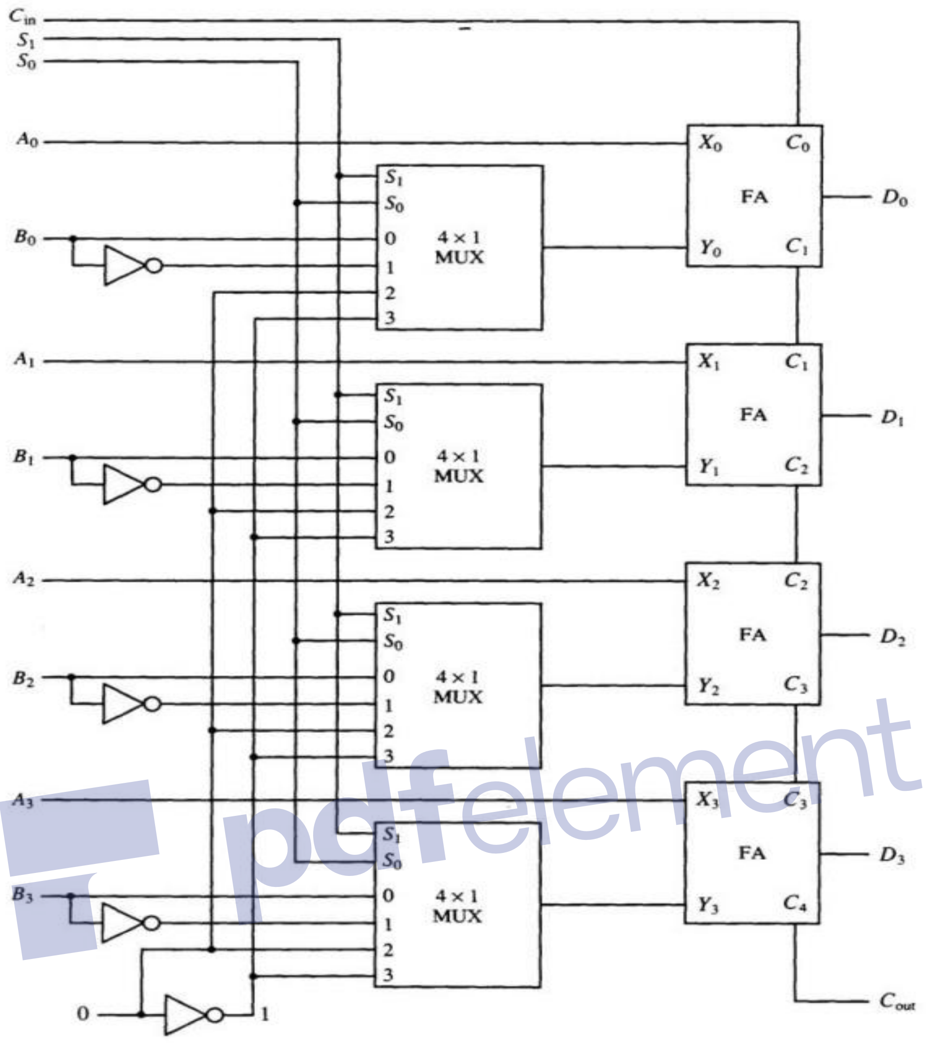


Figure 5 : 4-bit arithmetic circuit

3.3.2.3 Logical Microoperation

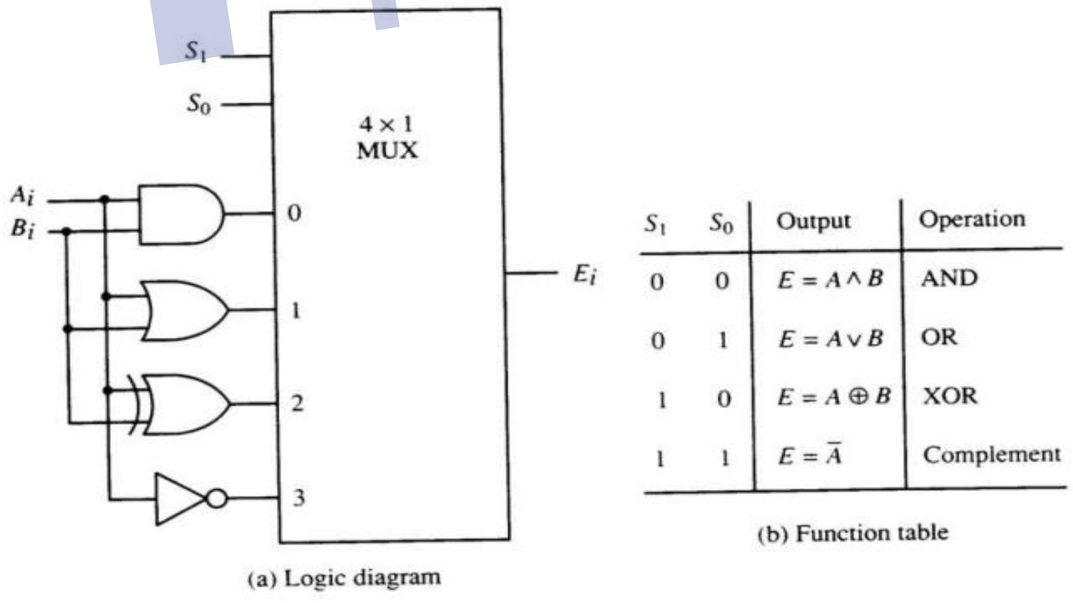
Logic micro operation specifies binary operations on the strings of bits in registers. Logic micro operations are bit-wise operations, i.e., they work on the individual bits of data. These are useful for bit manipulations on binary data and also useful for making logical decisions based on the bit value. There are many different logic functions that can be defined over two binary input variables. However, most systems only implement four of these: ***AND, OR, XOR, Complement/NOT***.

The others can be created from combination of these. The hardware implementation of logic micro operation requires the insertion of the most important gates like AND, OR, EXOR, and NOT for each bit or pair of bits in the registers.

Build a logical circuit to generate the four basic logic micro operations required:

- ***four gates*** (**AND, OR, XOR, NOT**) and
- ***a multiplexer***.

The two selection lines of the multiplexer selects one of the four logic operations available at one time. The circuit shows one stage for bit "i" but for logic circuit of n bits the circuit should be repeated n times but with one remark; the selection pins will be shared with all stages.



S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	Complement

Figure 6 Simple Logic circuit

3.3.2.4 Shift Microoperations

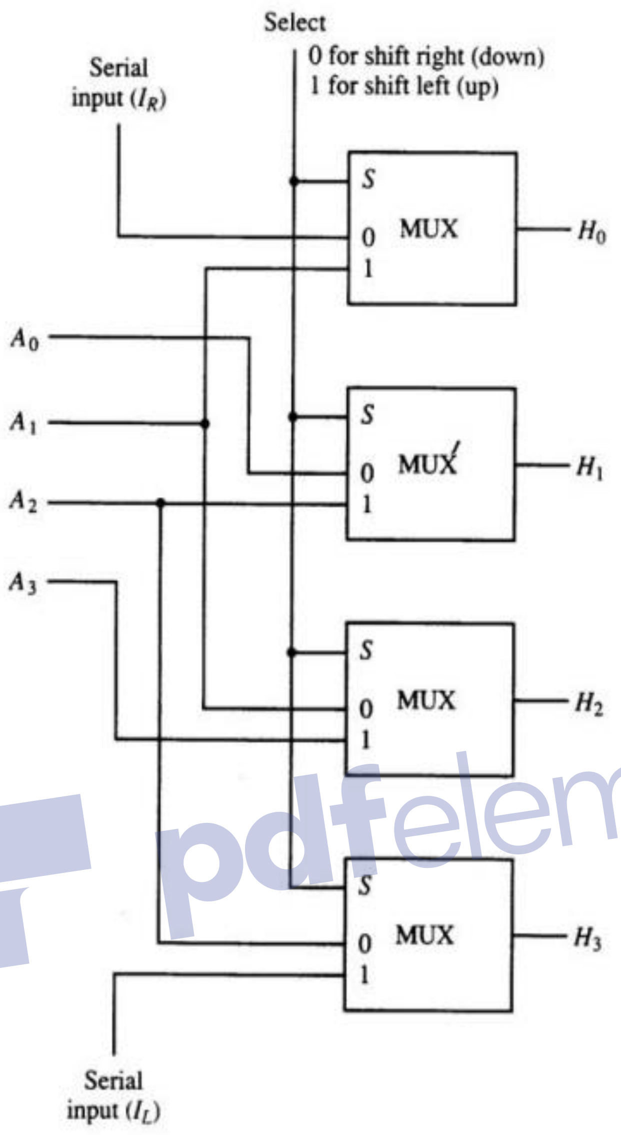
Shift micro-operations are used for **serial transfer** of data beside they are used in conjunction with arithmetic, logic, and other data processing operations. There are **3 types** of shift micro operations.

- **Logical shift** : Logical shift is one that transfers 0 through the serial input
- **Circular shift** : The circular shift rotates of the register around the two ends without loss of information
- **Arithmetic shift**: Arithmetic shift is a micro-operation that shifts a signed binary number to the left or right. Arithmetic shift must leave sign bit unchanged.

We can implement Shift microoperation using:

- Use **bidirectional shift register** with parallel load, In that case we need two clocks pulse one to **load** the value and another to **shift**.
- Another solution which is **more efficient** is to implement the shift operation with ***combinational circuits*** (combinational circuit will construct using multiplexers.)

Figure 9- below represents 4-bit combinational circuit shifter



Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Figure 7: 4-bit combinational circuit shifter

3.3.3 Arithmetic Logic Shift Unit

Instead of having individual registers performing micro-operations directly, computer systems employ a ***number of storage registers connected to a unit*** called **Arithmetic Logic Unit (ALU**). This unit has ***2 operands input ports*** and ***one output port*** and a ***number of select lines*** to help in ***selecting different operations***. The ALU is made of ***combinational circuit*** so that the entire register transfer operation ***from the sources to the destination is performed in one clock cycle***. The arithmetic, logic, and shift circuits (implemented previously) will be combined in one ALU with common selection inputs. Simple stage (bit) of ALU with its table is shown blow (figure 10). The arithmetic and logic units will select their operations simultaneously when $\mathbf{S_0}$ and $\mathbf{S_1}$ are applied; while $\mathbf{S_2}$ and $\mathbf{S_3}$ will select one of those unit outputs or a shift left bit stage or shift right bit stage. The circuit shown provides **8 arithmetic operations, 4 logic operations, and 2 shift operations.**

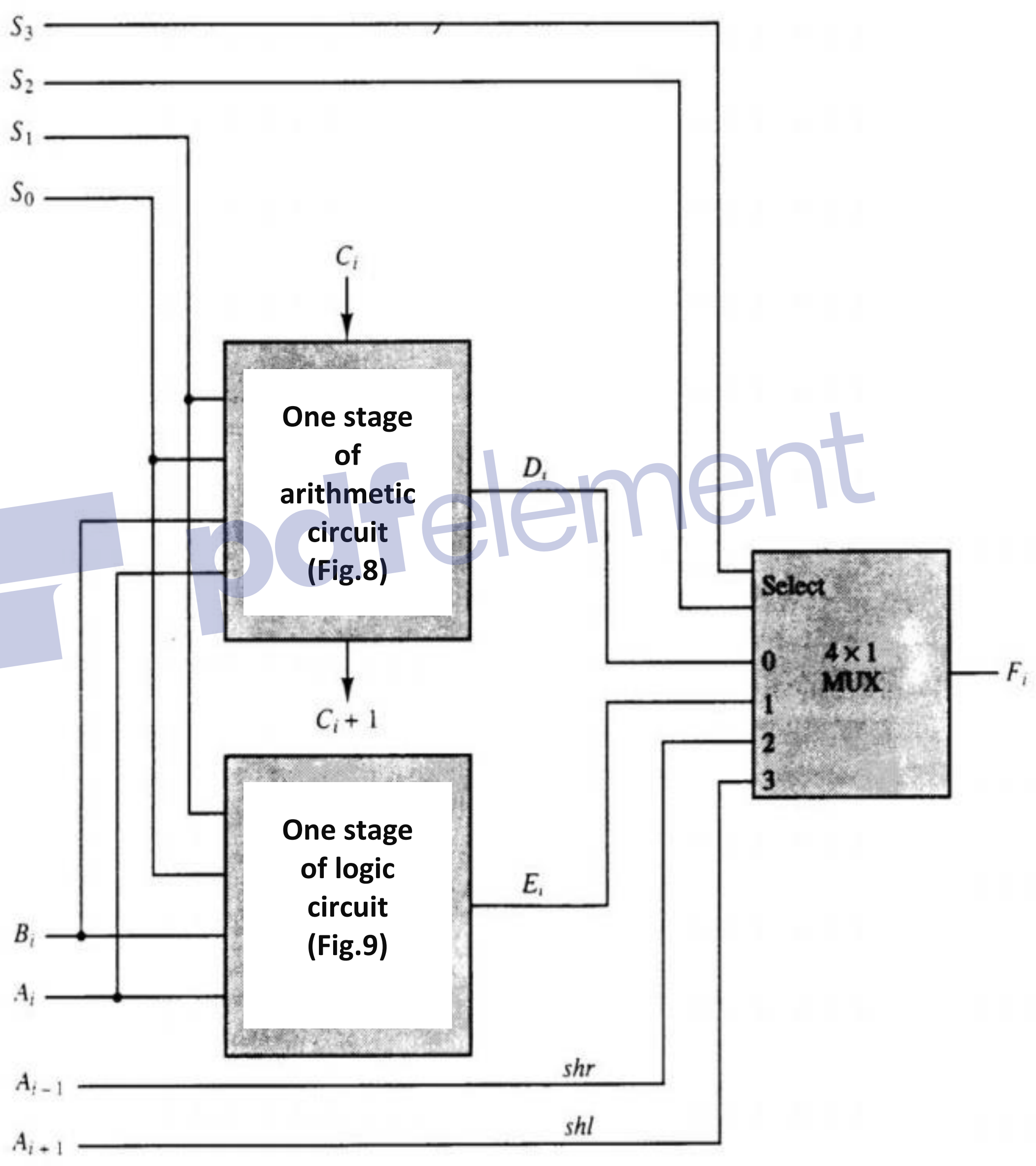


Figure 8 simple Arithmetic and logic unit

Table 2 Function table for Arithmetic -Logical (AND SHIFT) Unit

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F = \text{shr } A$	Shift right A into F
1	1	×	×	×	$F = \text{shl } A$	Shift left A into F