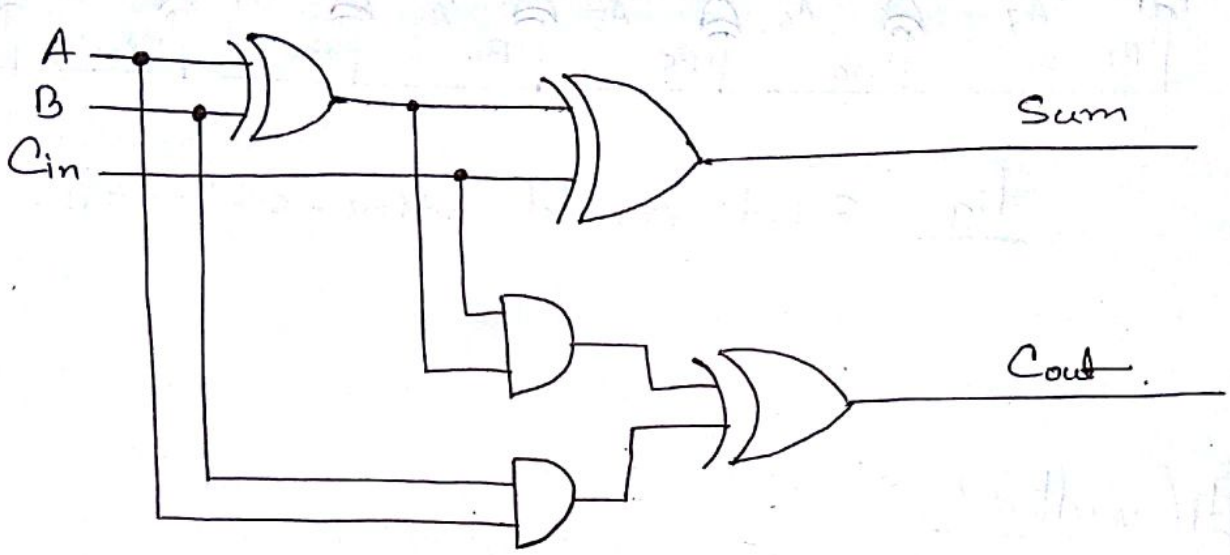
Chapter: 4 → Data path Design

8 bit adder-Subtractor. (Parallel): (4)

The circuit of full adder:



Truth table for full adder and subtractor :-

A	B	C	Full adder		Full subtractor	
			Sum	Carry	Diff.	Borrow
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	1	0	0
1	1	1	1	1	1	1

⇒ Logic diagram:

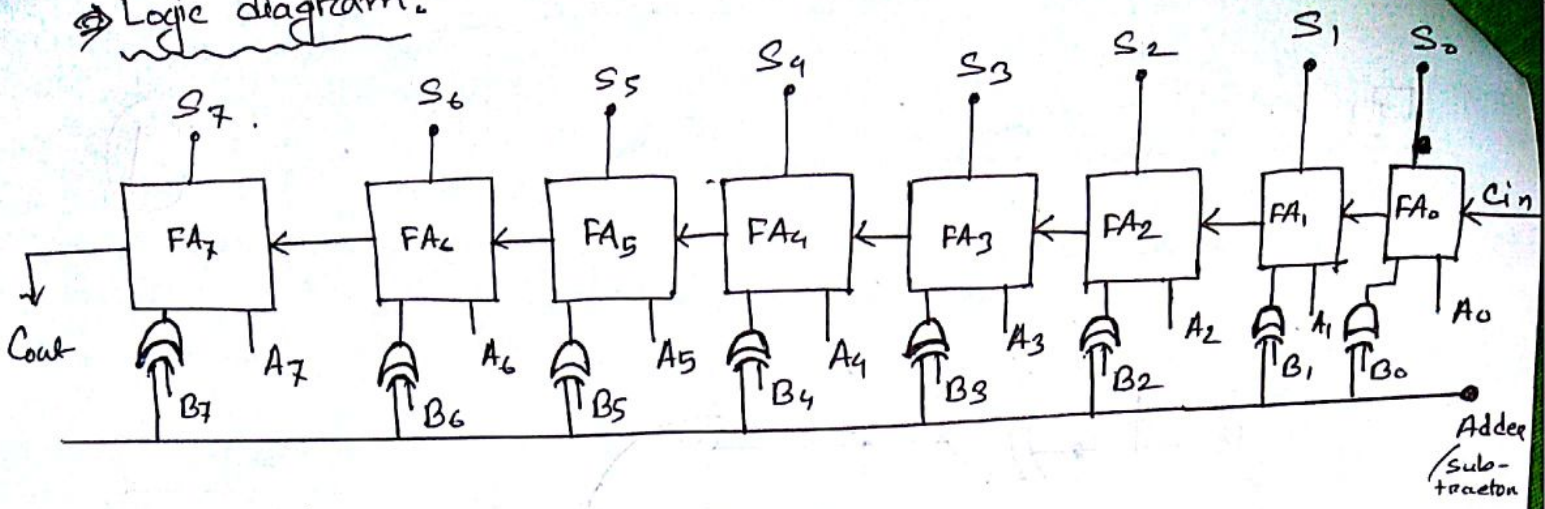


fig: 8 bit parallel adder - subtractor.

Overflow:

⇒ When the result of an arithmetic operation exceeds the standards word size n, overflow occurs.

Ex: $Z = X + Y = 11101011 + 00101010 = \overset{\textcircled{1}}{\downarrow} 00010101$

overflow is indicated by a flag bit V in operations involving ~~sing~~ signed numbers: this flag is found in CPU status registers.

⇒ The overflow condition is specified by the logic expression:

$$V = \bar{x}_{n-1}\bar{y}_{n-1}c_{n-2} + x_{n-1}y_{n-1}\bar{c}_{n-2} \quad \text{——} (1)$$

Now c_{n-1}, the carry output signal from the sign position, is defined by $(x_{n-1}y_{n-1} + x_{n-1}c_{n-2} + y_{n-1}c_{n-2})$ from which it follows that,

$$V = c_{n-1} \oplus c_{n-2}. \quad \text{——} (11)$$

Either (i) or (ii) can be used to design overflow ditection logic for 2's complement addition/subtraction.

High speed adders (Carry look ahead adder):

One approach is to compute the input carry needed by stage i directly from carry like signals obtained from all the preceding stages $i-1, i-2, \ldots, 0$ rather than waiting for the normal carries to ripple slowly from stage to stage. Adders that use this principle are called carry-lookahead adders.

in full adder, the output line C_i will replacing by two auxilary signals called g_i and P_i or generate and propagate.

$$g_i = x_i y_i \quad , \quad P_i = x_i + y_i$$

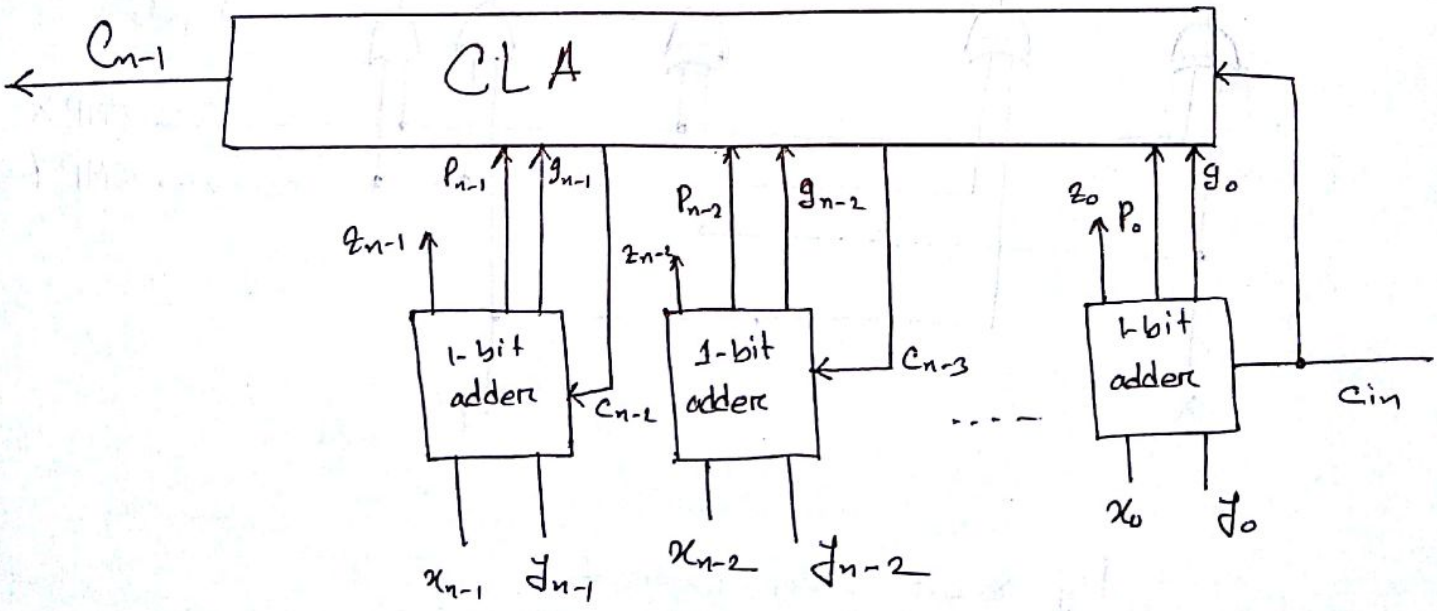


fig: Overall structure of carry-lookahead adder.

v.v2

Design of A complete 2's complement adder/-Subtractor:

We will design a two's-complement adder-subtracter that computes the three quantities, X+Y, X-Y, Y-X as well as overflow and zero flags. The design goal is to minimize the number of gates used; operating speed is not of concern.

The overflow flag is defined by, $V = C_{n-1} \oplus C_{n-2}$ and is realized here by an XOR gate. The zero detection requires access to all the sum outputs and poses no special problems.

$$\text{Zero}, \quad Z = \overline{Z_3 + Z_2 + Z_1 + Z_0}$$

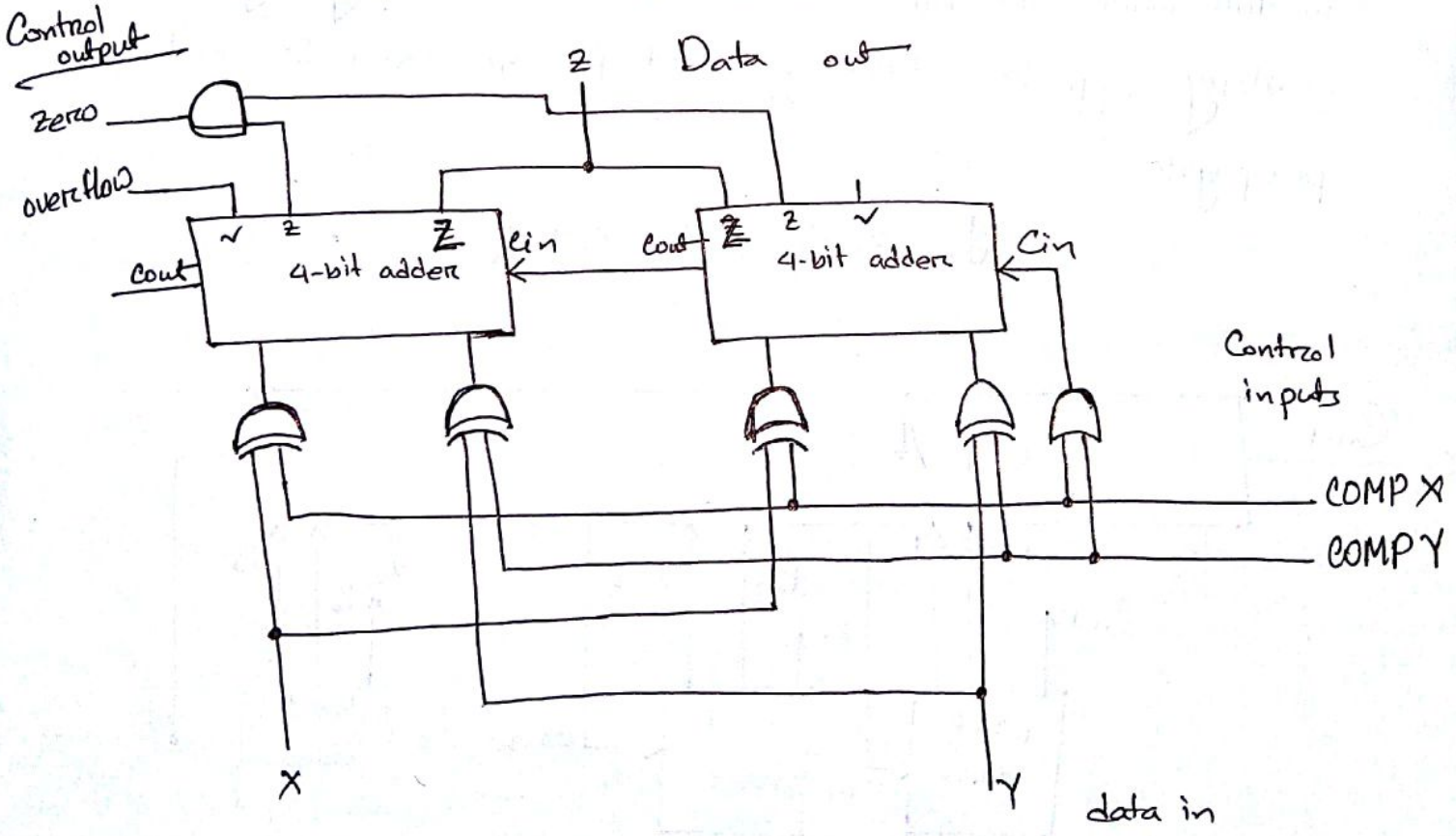


fig: 8-bit adder-subtractor.

It contains two 4-bit adders of the type in figure, linked by their carry lines. Two lines COMPX and COMPY control the XOR gates that change X and Y to $\bar{X}$ and $\bar{Y}$.

The OR gate sets the adder's carry-in line to 1 during Subtraction.

A two-input AND gate combines the two z outputs to produce the zero flag, which is 1 if and only if entire 8-bit result Z=0.

Three of the four signal combinations on COMPX and COMPY control lines implement the desired three arithmetic functions.

The fourth combination 11 implements the sum $\bar{X}+\bar{Y}=1$

Multiplication

① Fixed-point multiplication requires substantially more hardware than fixed-point addition.

② Multiplication is usually implemented by some form of repeated addition.

③ The main operations involved are shifting and addition.

$$P_{i+1} = P_i + x_i 2^i Y \quad \text{——— ①}$$

Ex:

```
        1 0 1 0  → Multiplicand, Y
        1 1 0 1  → Multiplier, X
-------------------------------------------
0 0 0 0 0 0 0 0  → P0 = 0
      1 0 1 0    → x0 2^0 Y
-------------------------------------------
0 0 0 0 1 0 1 0  → P1 = P0 + x0 2^0 Y
    0 0 0 0      → x1 2^1 Y
-------------------------------------------
0 0 0 0 1 0 0 1 0 → P2 = P1 + x1 2^1 Y
  1 0 1 0         → x2 2^2 Y
-------------------------------------------
0 0 1 1 0 0 1 0  → P3 = P2 + x2 2^2 Y
  1 0 1 0         → x3 2^3 Y
-------------------------------------------
1 0 0 0 0 0 1 0  → P4 = P3 + x3 2^3 Y.
```

fig: The multiplication of two binary numbers modified for machine implementation.

⇒ $2^i Y$ is equivalent to Y shifted i positions to the left.

Two's-complement multiplier:

⇒ A conceptually simple approach to two's complement multiplication is to negate all negative operands at the beginning, perform unsigned multiplication on the resulting (positive) numbers, and then negate the result if necessary. Two's complement negation for an integer, $X = x_{n-1} x_{n-2} \cdots x_1 x_0$ is specified by,

$$-X = \bar{x}_{n-1} \bar{x}_{n-2} \cdots \bar{x}_1 \bar{x}_0 + 000\ldots01 \quad (\text{modulo } 2^n).$$

Booth's algorithm:

Definitions: Booth's algorithm employs both addition and subtraction, but it treats positive and negative operands uniformly - no special actions are required for negative numbers. Booth's algorithm can also be readily extended in various ways to speed up the multiplication process.

⇒ In Booth's approach two adjacent bits $x_i x_{i-1}$ are examined in each step. If $x_i x_{i-1} = 01$ then Y is "added" to the current partial product P_i, while if $x_i x_{i-1} = 10$, Y is "subtracted" from P_i. If $x_i x_{i-1} = 00$ or 11 then neigther addition or subtraction is performed.

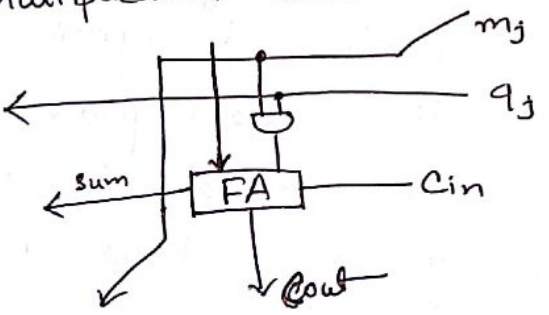
Array Implementation of the Booth Multiplication Algo.

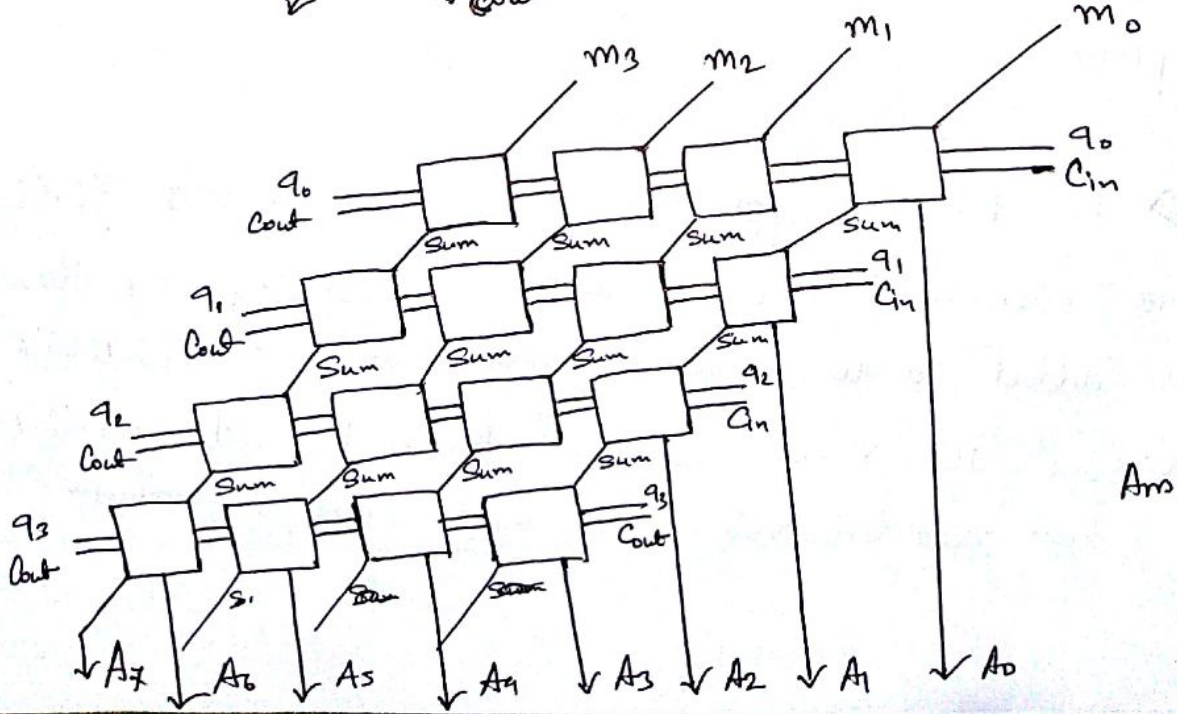
Implementing the Booth method by a combinational array requires a multifunction cell capable of addition, subtraction, and no operation (skip).

Ex: $(14)_{10} \times (10)_{10} = (140)_{10}$

```
  1 1 1 0  → Multiplicand (M)   (14)
  1 0 1 0  → Multiplier (Q)     (10)
-----------
    0 0 0 0
  0 1 1 0
0 0 0 0          → Partial product.
1 1 1 0
-----------
1 0 0 0 1 1 0 0  → (140)10 Final product.
```

Basic multiplication addition circuit.





Ans: A

ALU

Definition :

The various circuits used to execute data-processing instructions are usually combined in a single circuit called an' arithmetic-logic unit (ALU).

⇒ Simple ALUs that perform fixed-point addition and Subtraction, as well as word based logical operations, can be realized by combinational circuits.

⇒ ALU is divided into two units, an arithmetic unit (AU) and a logic unit (LU). Some processors contain more than one ALU - for example, one for fixed-point operations, and another for floating-point iterations.

Features of Combinational ALU :

① The simplest ALUs combine the functions of a 2's complement adder-subtracter with those of a circuit that generates word-based logic functions of the form f(X,Y).

② It can thus implement most of a CPU's fixed-point data-processing instructions.

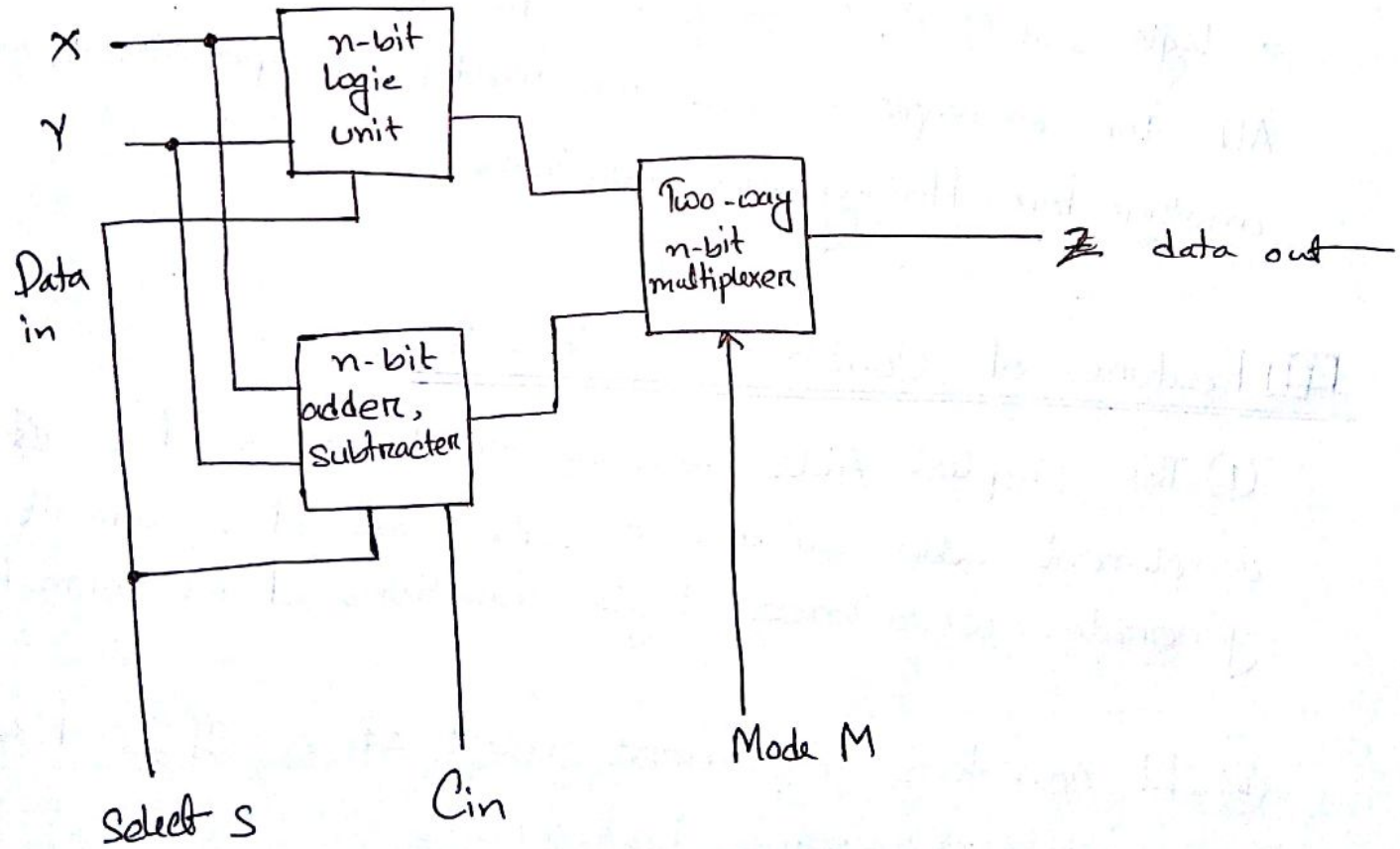
③ ALU that has separate subunits for logical and arithmetic operations.

④ The particular class of operation (logical and arithme
to be performed is determined by a "mode" control
line M attached to a two-way multiplexer.

⑤ The specific operation performed by the desired subunit
is determined by a "select" control line S.

⑥ The ALU's logical operations are performed bitwise.

(4) Features of Sequential ALU:



Sequential ALU:

Complete ALUs are usually constructed from low-cost sequential circuit where add and subtract each take one clock cycle, while multiplication and division are multicycle operations.

ALU expansion:

It is quite feasible to manufacture an entire sequential ALU for fixed-point m-bit numbers on a single IC chip. Moreover, the ALU can easily be designed for expansion to handle operands of size $n = km$, or indeed any word size $n > m$, in two ways:

① Spatial expansion.
② Temporal expansion.

① Spatial expansion:

Connect k copies of the m-bit ALU in the manner of a ripple-carry adder to form a single ALU capable of processing km-bit words directly. The resulting array-like circuit is said to be "bit sliced" because each component ALU concurrently processes a separate "slice" of m-bits from each km-bit operand.

(4) Temporal expansion: Use one copy of the m-bit ALU slice in the manner of a serial adder to perform an operation on km-bit words in k consecutive steps (clock cycle). In each step the ALU processes a separate m-bit slice of each operand. This processing is called "multicycle" or "multiple-precision" processing.

Organization of 16 bit ALU using 4-bit slice:

The data buses and register files of the individual slices are effectively juxtaposed to increase their size from 4 to 16 bits.

The control lines that select and sequence the operations to be performed are connected to every slice so that all slices execute the same actions in lockstep with one another.

Each slice thus performs the same operations on a different 4-bit part (slice) of the input operands and produces only the corresponding part of the results.

The required control signals are derived from an external control unit, which can be hardwired or microprogrammed. Certain operations require information to be exchanged between slices.

A multicycle implementation of the 16-bit ALU of the figure would require the basic 4-bit ALU to store internally all the information that needs to be exchanged between slices. Add and shift operations require only modest changes like extra flip-flops to store the output carry and shift signals.

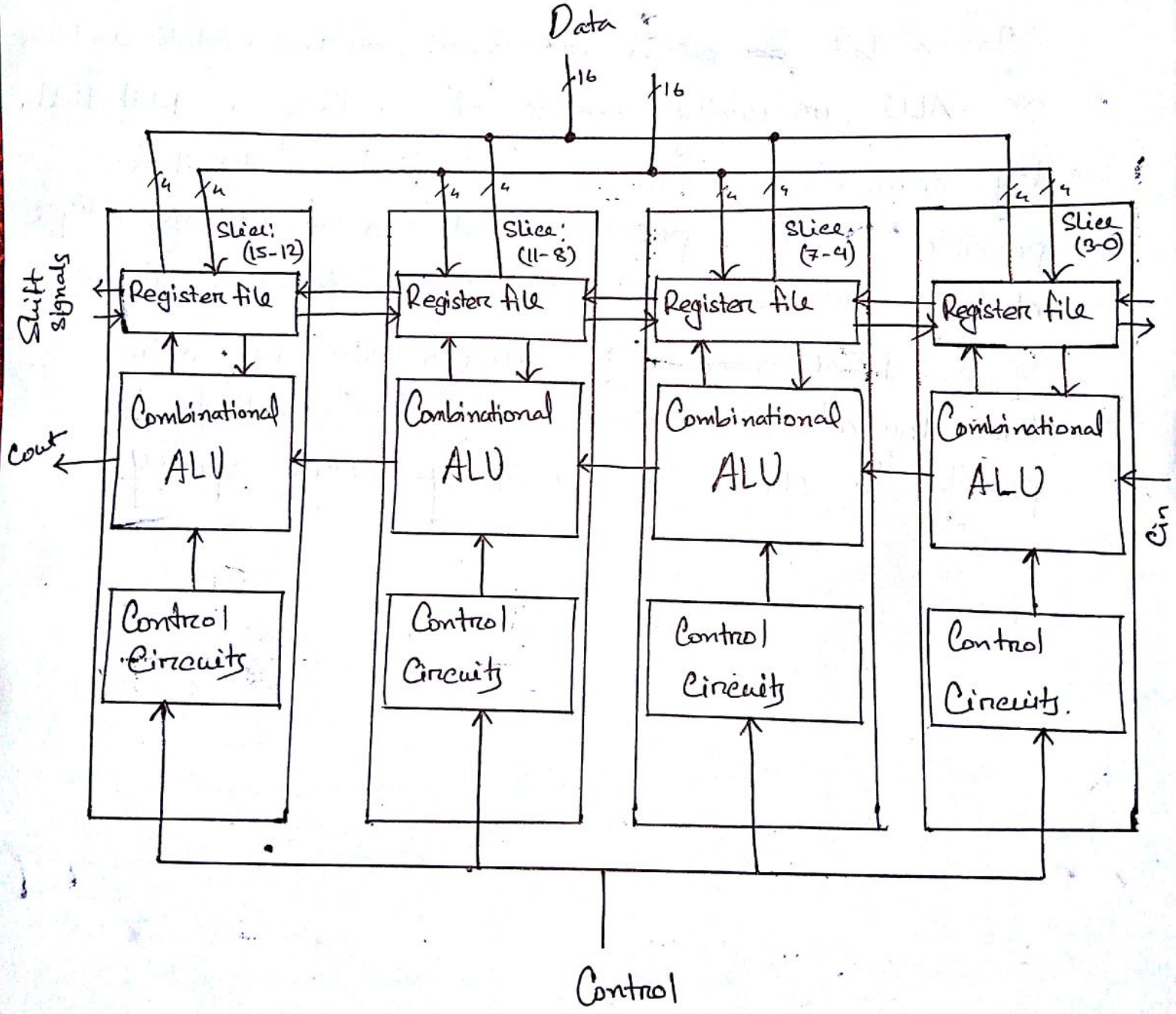


fig: 16 bit ALU composed of four 4-bit slices.

Bit slicing:

Bit slicing is a method of combining processor modules to multiply the word length. Bit slicing was common with early processors, notably the AMD 2900 series.

Concept of Bit sliced ALU:

In a bit sliced processor, each module contains an ALU, usually capable of handling a 4-bit field. By combining two or more identical modules, it is possible to build a processor that can handle any multiple of this value, such as 8 bits, 12 bits, 16 bits, 20 bits and so on. Each module is called a slice. The control line for all the slices are connected effectively in parallel to share the processing "work" equally.