Datapath Functional Units

Contents

Introduction

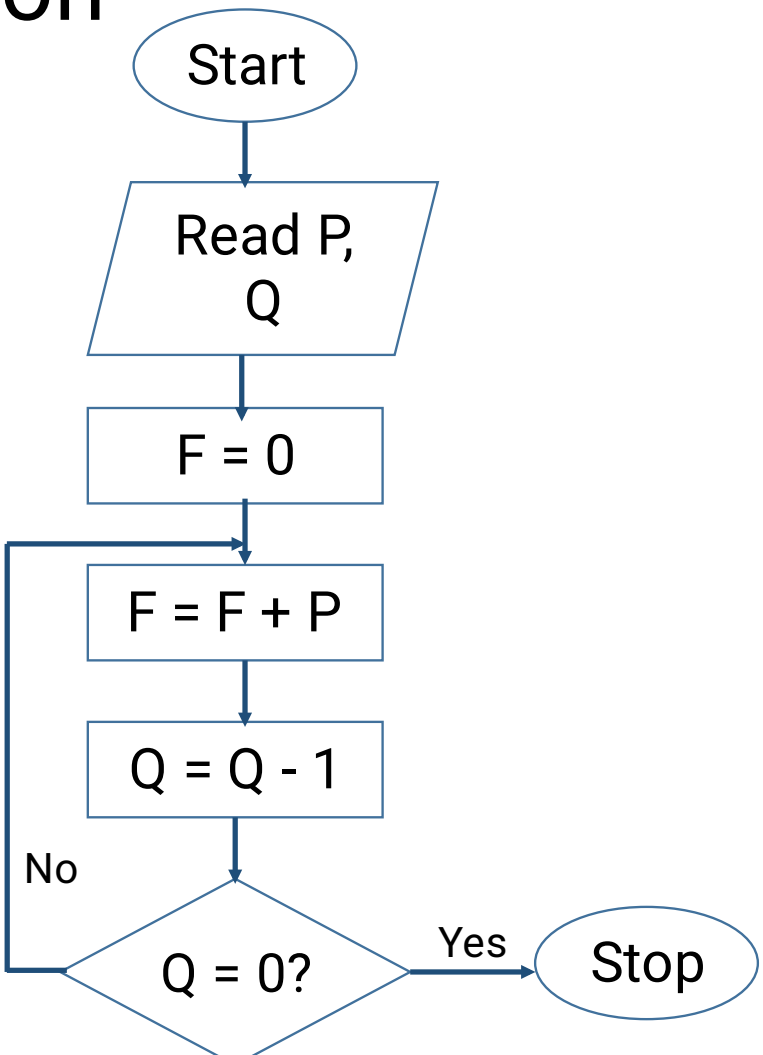
- Data path
  - Consists of functional units where all computations are carried out
    - Ex: Registers, multiplexers, bus, adders, multipliers, counter
- Control path
  - Consists of FSM and provide control signals to the data path in proper sequence
  - With the help of control signals various operations are carried out by the data path
  - Also takes inputs from the data path regarding status information

Introduction

- Illustrate data and control path
  - P = Q + R
  - S = P - R

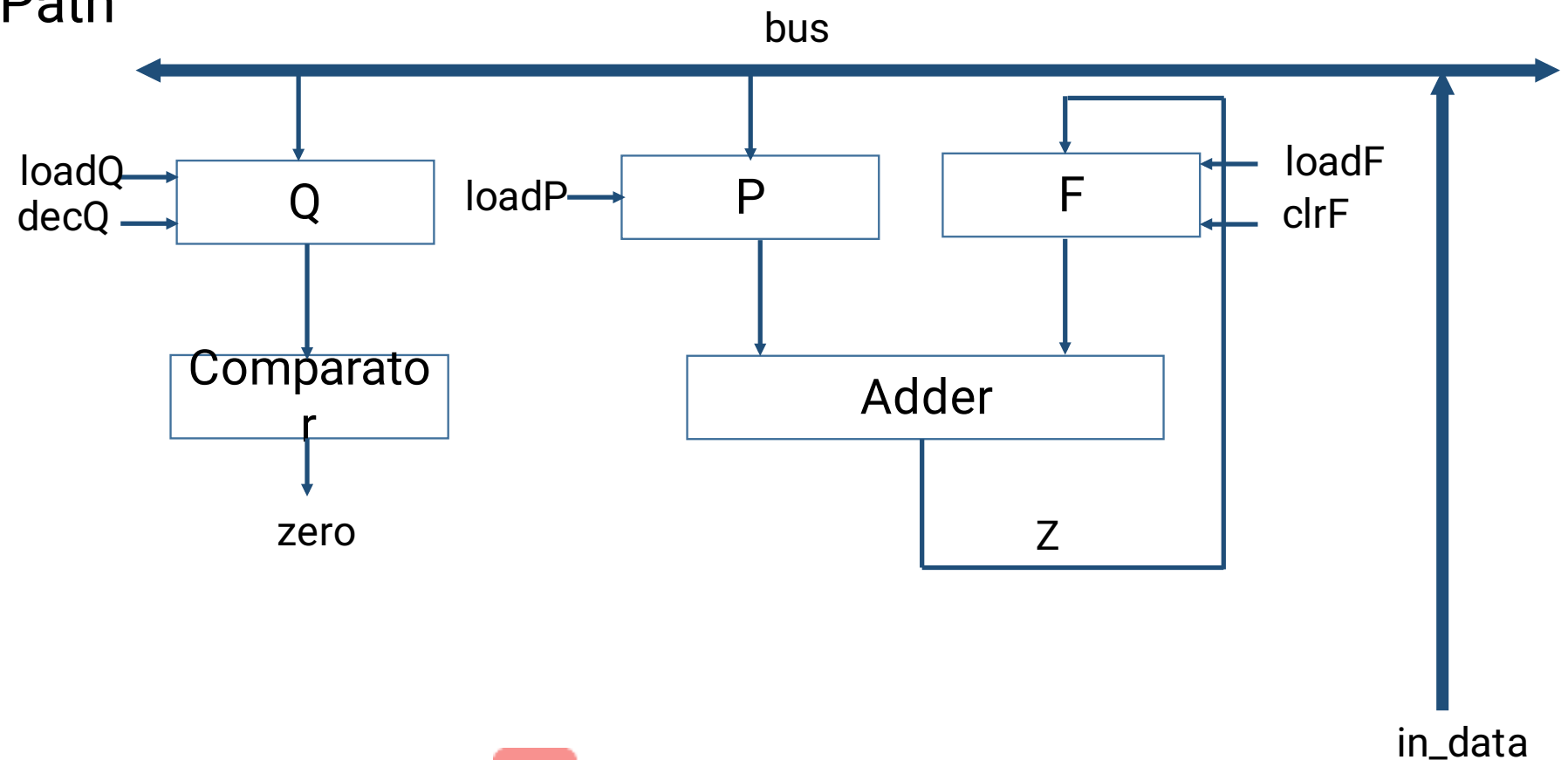
Multiplication by repeated addition

- Illustrate (P × Q) by repeated addition
- Assumption, Q is not zero
- Identify the functional blocks
- Design the FSM to implement the algorithm with the help of data path



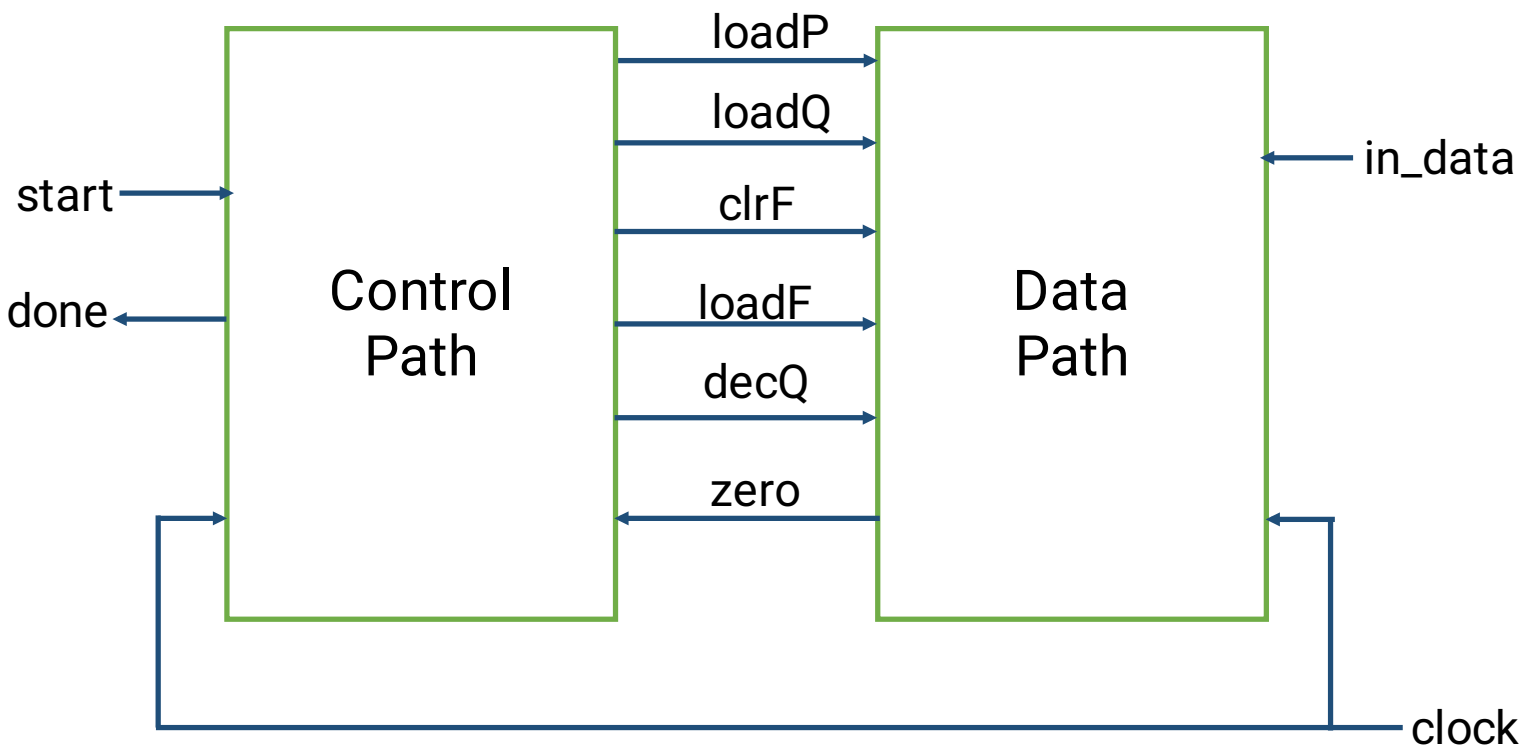
Multiplication by repeated addition

- Data Path



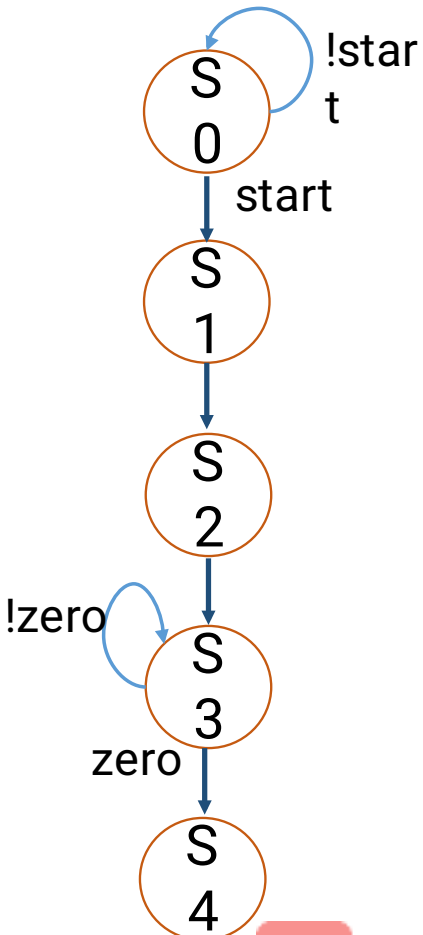
Multiplication by repeated addition

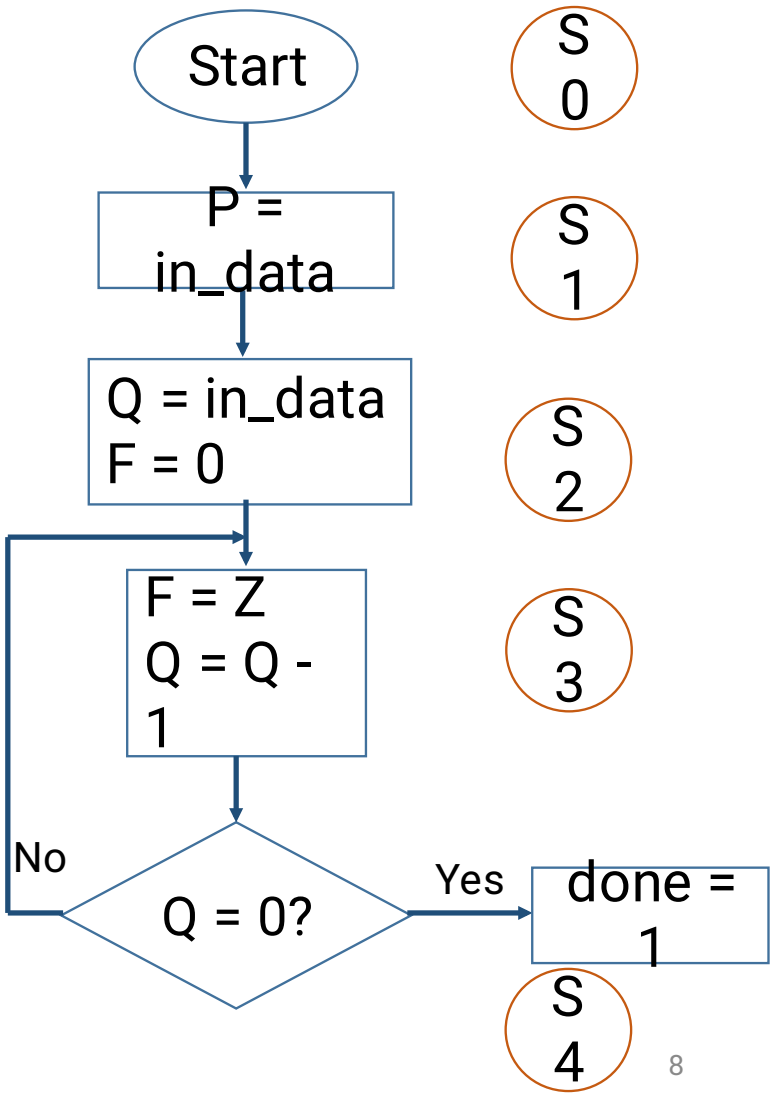
- Block Diagram



Multiplication by repeated addition

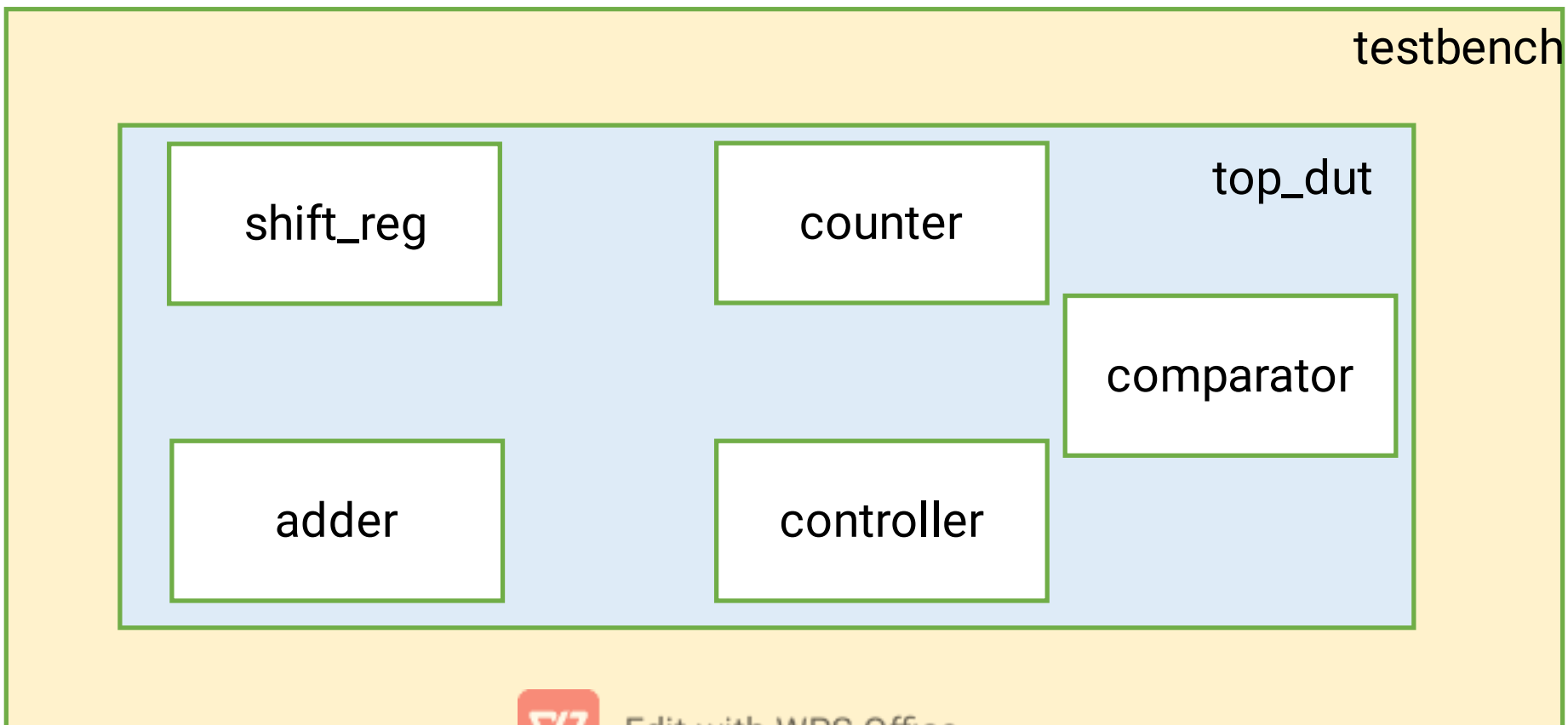
- Control Path





Multiplication by repeated addition

- Write HDLs and simulate

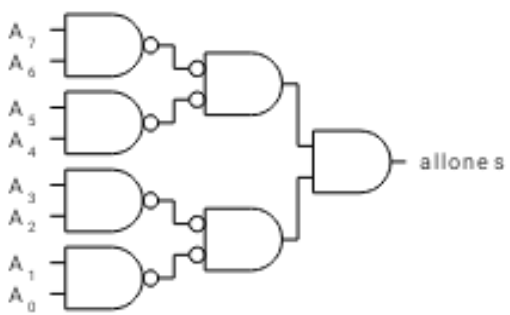


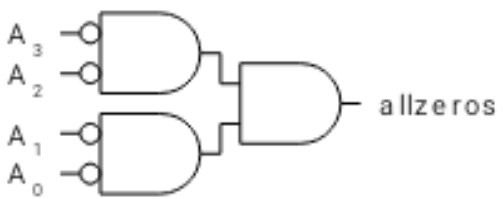
Comparators

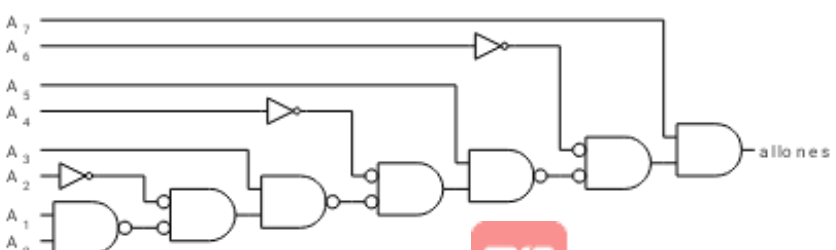
- 0’s detector: A = 00…000
- 1’s detector: A = 11…111
- Equality comparator: A = B
- Magnitude comparator: A < B

1’s & 0’s Detectors

- 1’s detector: N-input AND gate
- 0’s detector: NOTs + 1’s detector (N-input NOR)

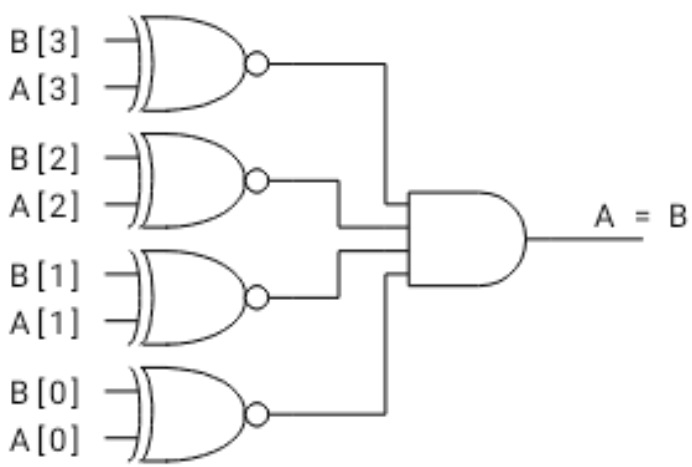






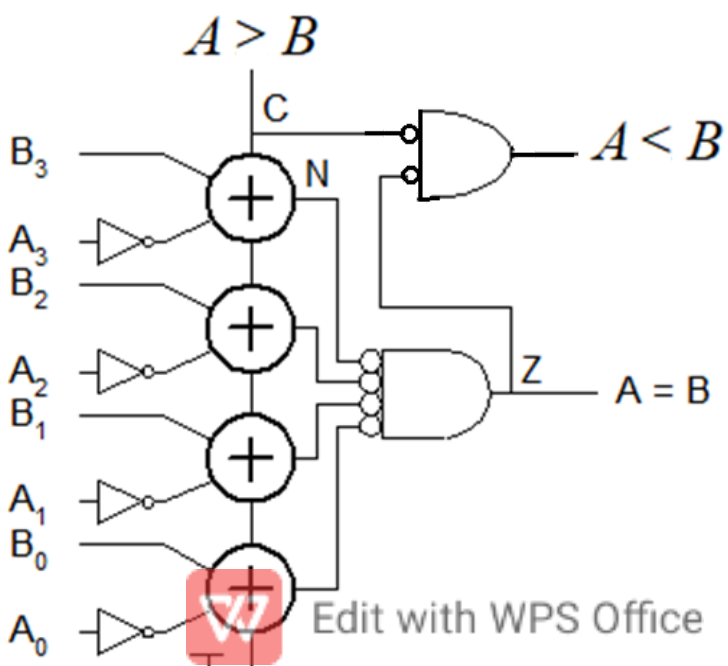
Equality Comparator

- Check if each bit is equal (XNOR, an equality detector gate)
- 1’s detect on bitwise equality



Magnitude Comparator

- Compute B-A and look at sign
- B-A = B + ~A + 1
- For unsigned numbers, carry out is sign bit



Detecting overflows

1) When the result of the operation is outside the representable range an overflow occurs.

2) Overf bws can only occur when the sign of the two operands is the same and if the sign of the result is different from the sign of the operands.

- Recall that the MSB represents the sign.
  - x_{n-1}, y_{n-1}, s_{n-1} represent the sign of operand *x*, operand *y* and result *s* respectively.
- Circuit to detect overf bw can be implemented bv the following logic expression:

$$Overflow = x_{n-1} y_{n-1} \bar{s}_{n-1} + \bar{x}_{n-1} \bar{y}_{n-1} s_{n-1}$$

Detecting overflows

3) It can also be shown that overflow occurs when the carry bits c_n and c_{n-1} are different.

Therefore, a simpler circuit for detecting overflow can be obtained by implementing the expression $c_n \oplus c_{n-1}$ with an XOR gate.

$$Overflow = c_n \oplus c_{n-1}$$

Detecting overflows

Example indicating overflow is occurred

1) +5 + +6

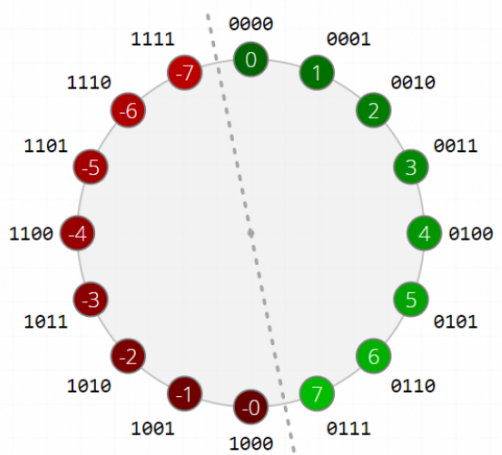
0101 + 0110 = 1011 overflow is occurred

2) +3 – (-7)

0011 – 1001 = 0011 + 0111 = 1010 overflow is occurred

Signed vs. Unsigned

- For signed numbers, comparison is harder
  - C: carry out
  - Z: zero (all bits of A-B are 0)
  - N: negative (MSB of result)
  - V: overflow (inputs had different signs, output sign ≠ B)



Considering B - A

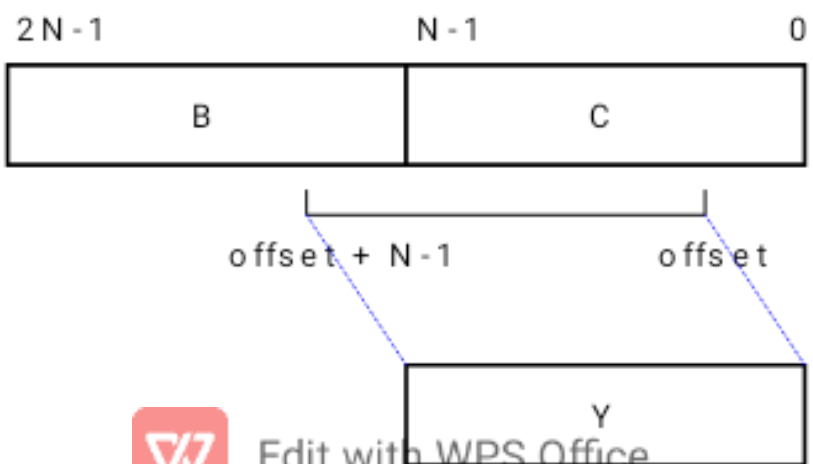
Relation	Unsigned Comparison	Signed Comparison
$A = B$	Z	Z
$A \neq B$	$\overline{Z}$	$\overline{Z}$
$A < B$	$C \cdot \overline{Z}$	$\overline{S} \cdot \overline{Z}$
$A > B$	C	S
$A \leq B$	C	$\overline{S}$
$A \geq B$	$\overline{C} + Z$	$S + Z$

Shifters

- Logical Shift:
  - Shifts number left or right and fills with 0’s
    - 1011 LSR 1 = 01011011 LSL1 = 0110
- Arithmetic Shift:
  - Shifts number left or right. Right shift sign extends
    - 1011 ASR1 = 1101 1011 ASL1 = 0110
- Rotate:
  - Shifts number left or right and fills with lost bits
    - 1011 ROR1 = 11011011 ROL1 = 0111

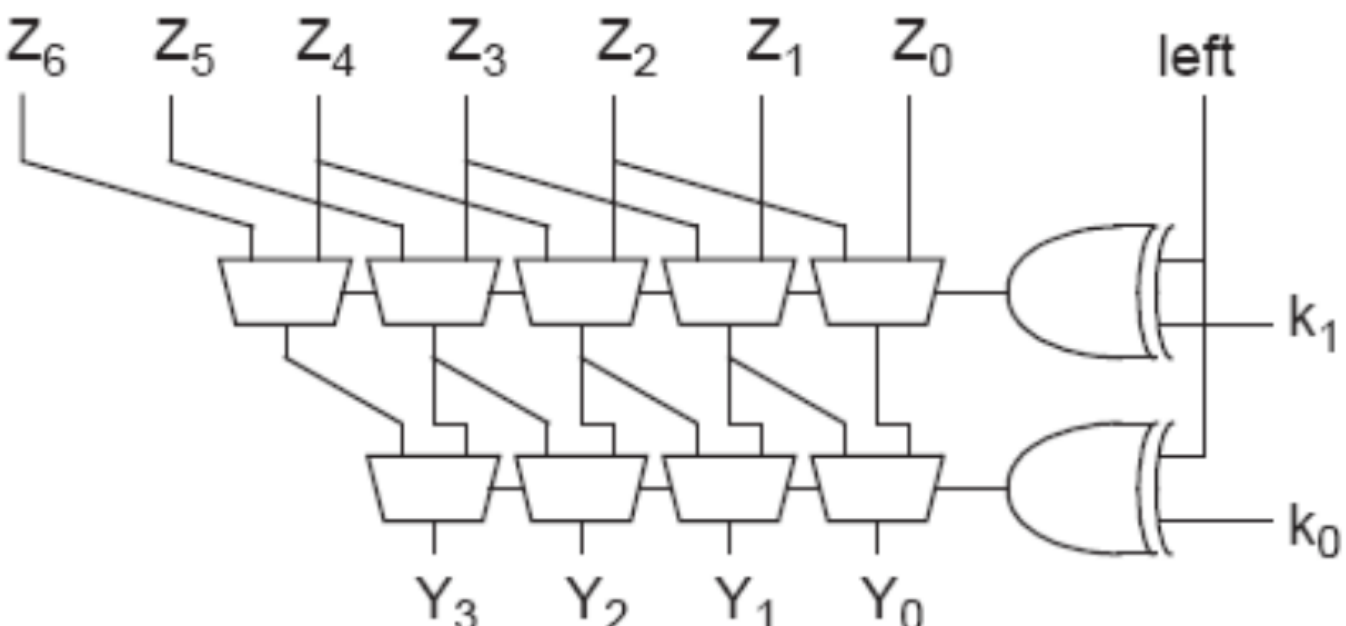
Funnel Shifter

- A funnel shifter can do all six types of shifts
- A funnel shifter creates a 2N – 1-bit input word Z from A then selects an *N*-bit field from this input word
- Selects N-bit field Y from 2N-bit input
  - Shift by k bits ($0 \leq k < N$)



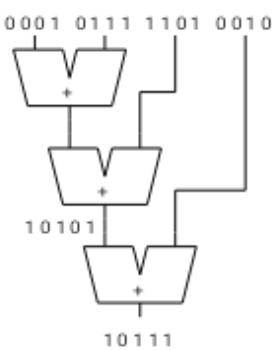
Funnel Shifter Design

- Log N stages of 2-input muxes
  - No select decoding needed



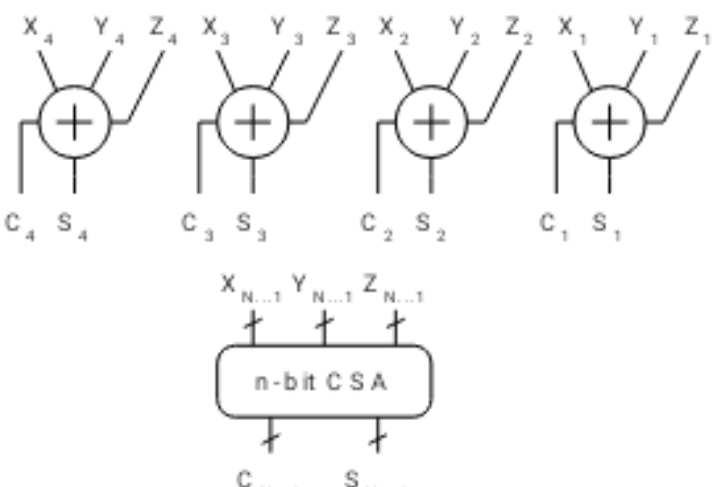
Multi-input Adders

- Suppose we want to add k N-bit words
  - Ex: 0001 + 0111 + 1101 + 0010 = 10111
- Straightforward solution: k-1 N-input CPAs
  - Large and slow



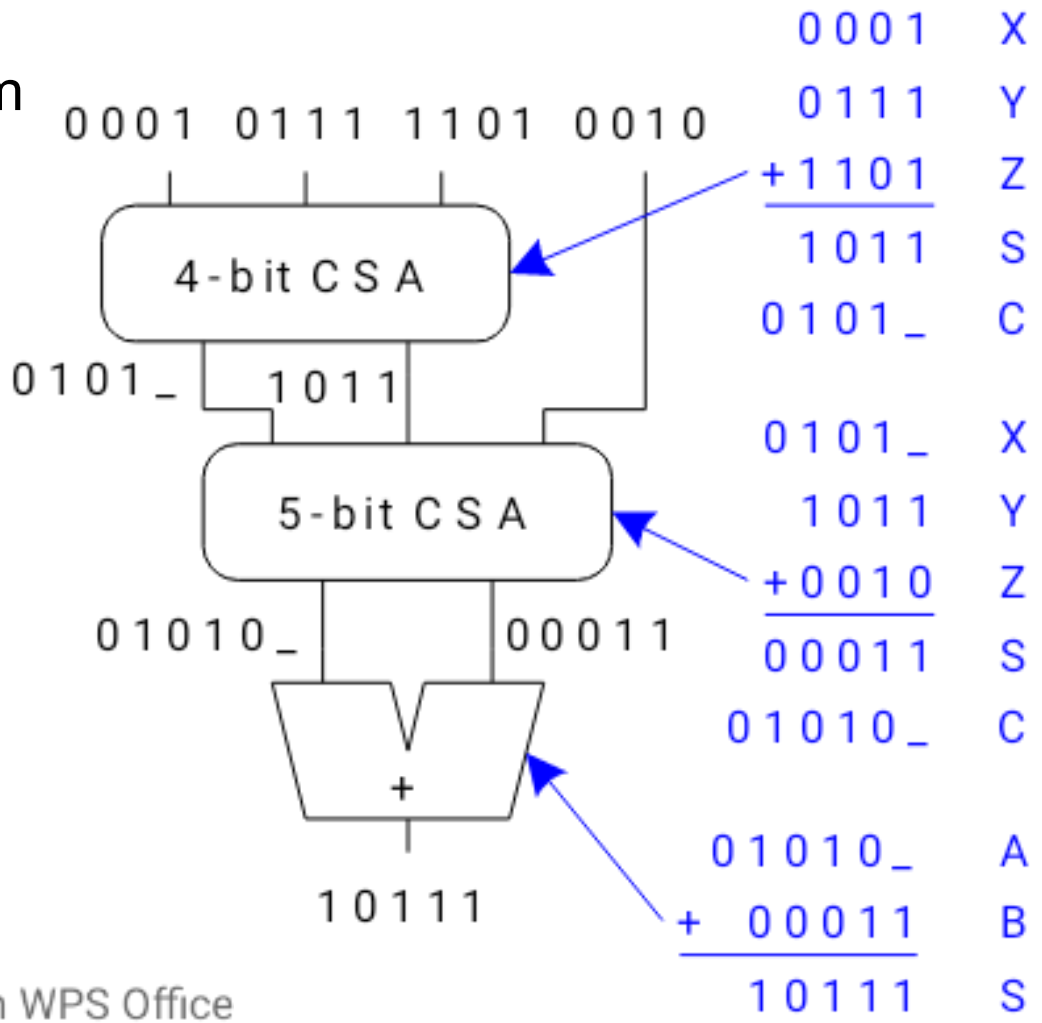
Carry Save Addition

- A full adder sums 3 inputs and produces 2 outputs
  - Carry output has twice *weight* of sum output
- N full adders in parallel are called *carry save adder*
  - Produce N sums and N carry outs



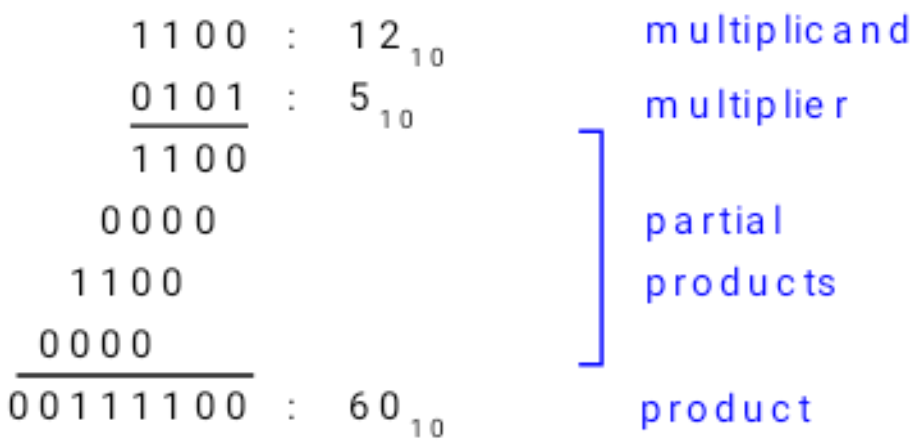
CSA Application

- Use k-2 stages of CSAs
  - Keep result in carry-save redundant form
- Final CPA computes actual result



Multiplication

- Example:

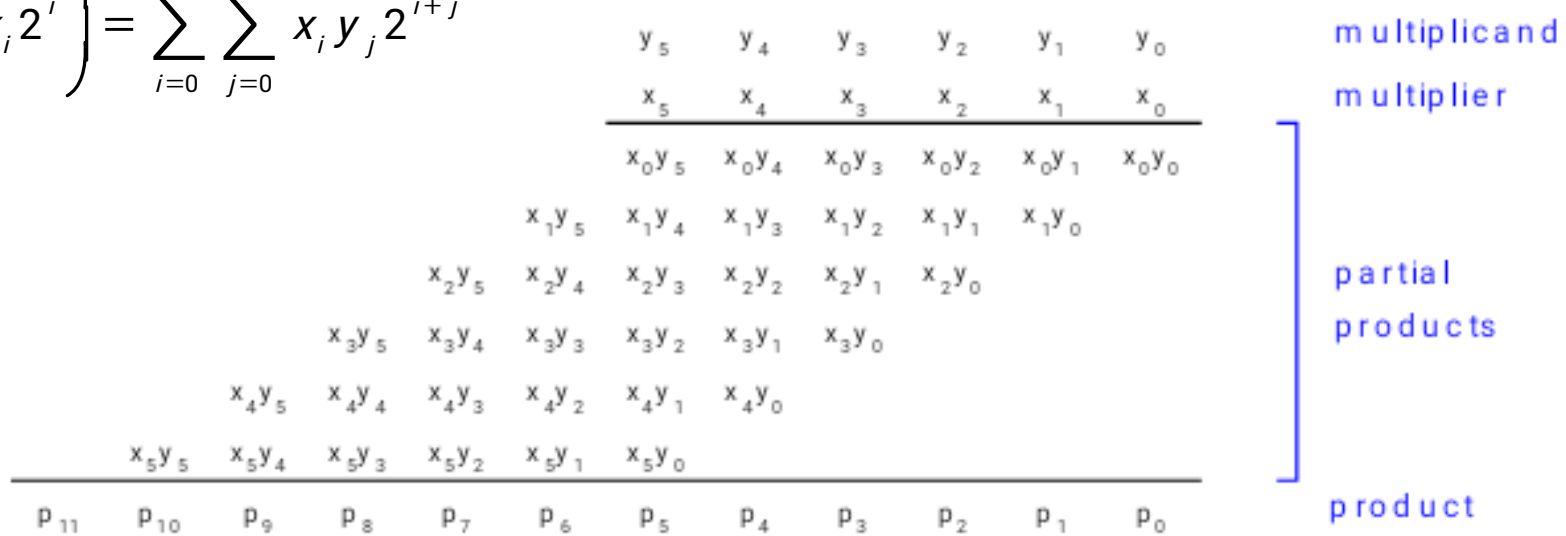


- M x N-bit multiplication
  - Produce N M-bit partial products
  - Sum these to produce M+N-bit product

General Form

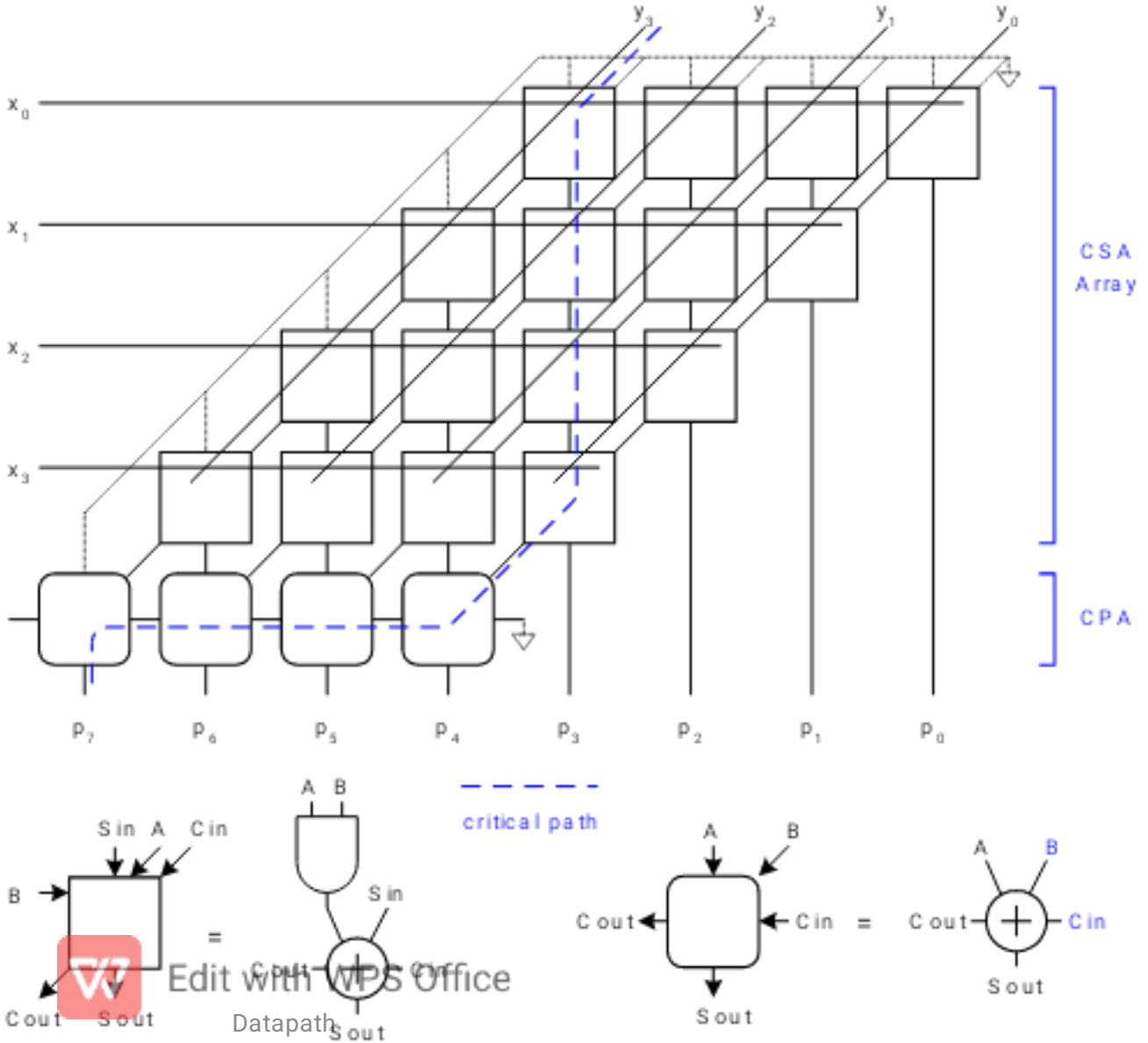
- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product:

$$P = \left(\sum_{j=0}^{M-1} y_j 2^j \right) \left(\sum_{i=0}^{N-1} x_i 2^i \right) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$



Array Multiplier

- Array multiplier using CSA and CPA



Divider

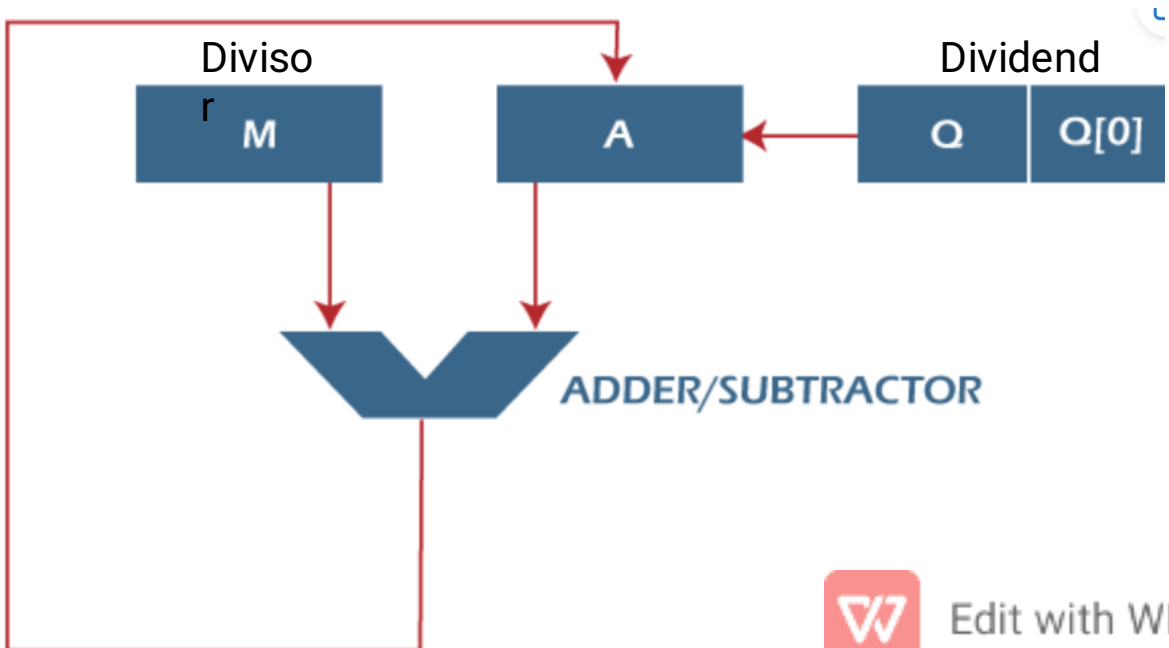
- Division algorithm – Two types
  - Fast Division Algorithms

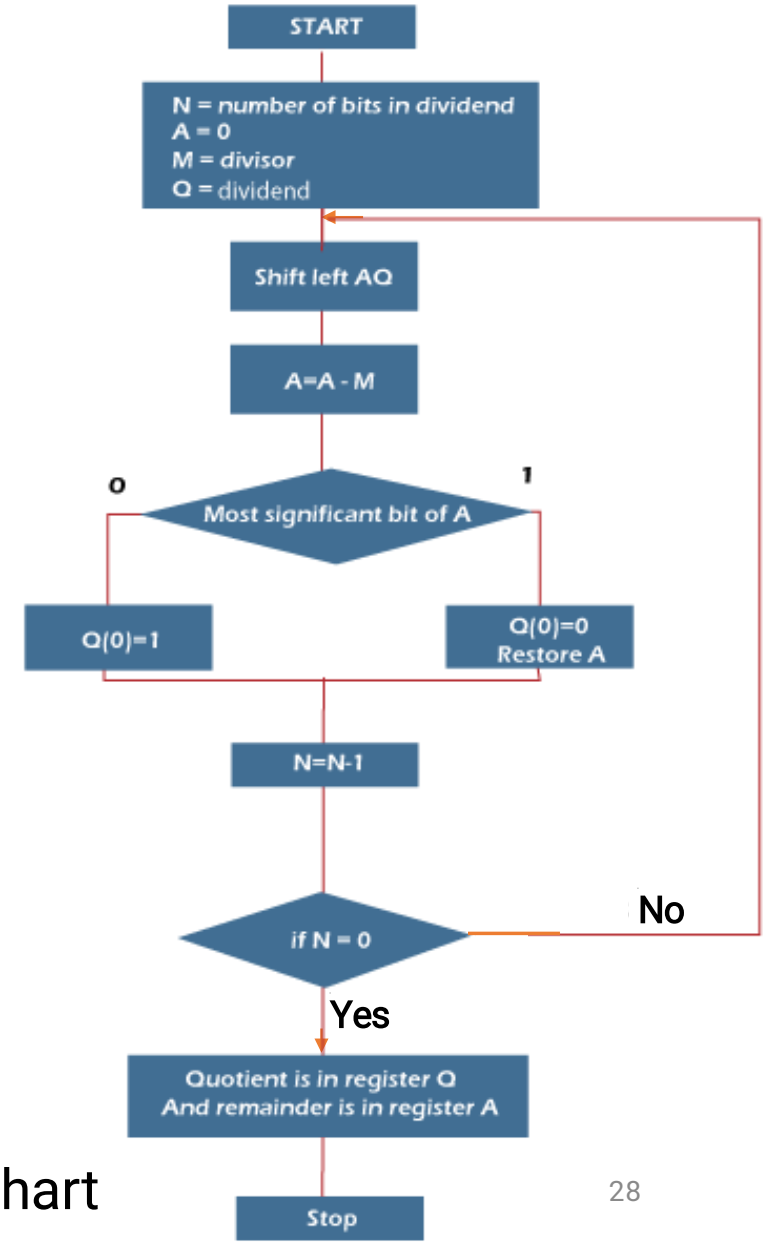
    Ex.: Goldschmidt algorithm, Newton-Raphson algorithm
  - Slow Division Algorithms

    Ex.: STR algorithm, **restoring algorithm**, non-performing algorithm, and the **non-restoring algorithm**

Restoring Division Algorithm

- Division of an unsigned integer.
- ‘Restoring’ term because value of register A will be restored after each iteration.





Flow Chart

- **Step 1:** The corresponding value will be initialized to the registers

  A ← 0, M ← Divisor, Q ← Dividend, and N ← number of bits in dividend.

- **Step 2:** Register A and register Q will be treated as a single unit, and the value of both the registers will be shifted left.

- **Step 3:** Value of M subtracted from A. The result of subtraction stored in A.

- **Step 4:** Now, check the MSB of A. If this bit of A is 0, then the LSB of Q set to 1. If the MSB of A is 1, then the LSB of Q set to 0, and restore the value of A that means it will restore the value of A before subtraction with M.

- **Step 5:** Value of N decremented. Here N is used as a counter.

- **Step 6:** If the value of N is 0, break the loop. Otherwise, go to step 2.

- **Step 7:** This is the last step. Q contains quotient, and A contains remainder.

Example: 11 / 3

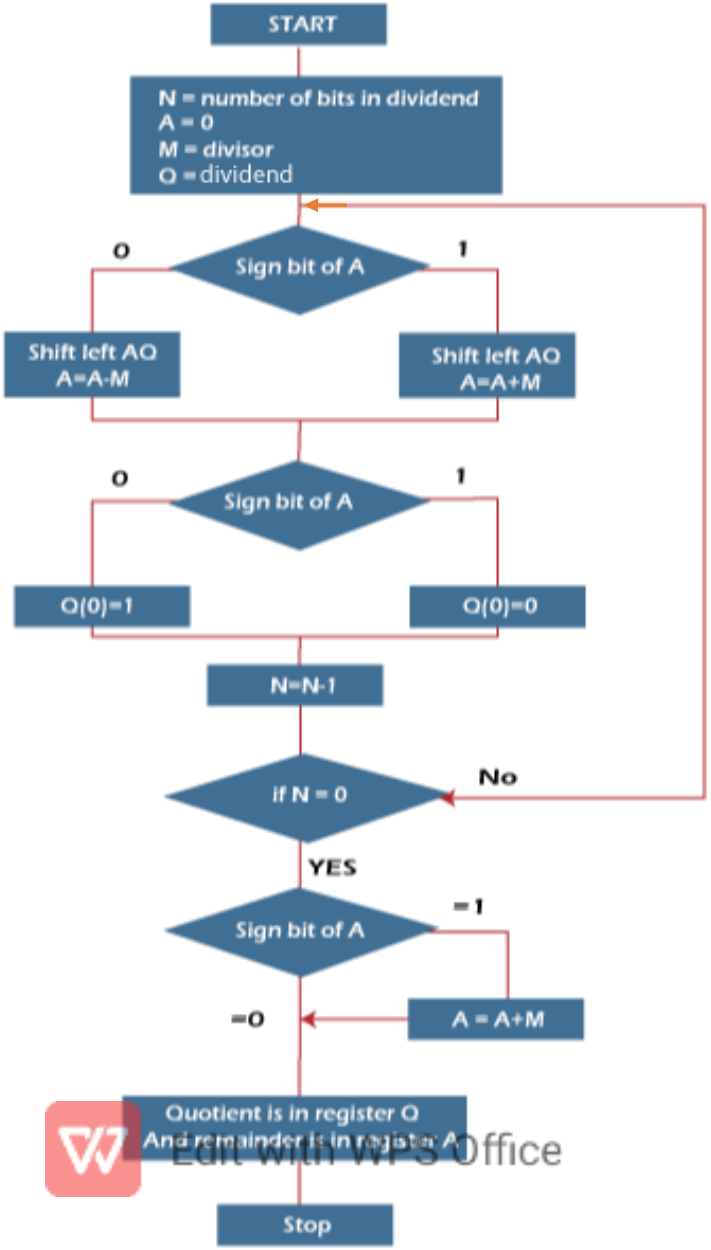
- We should not forget to restore the value of the MSB of **A**, which is 1.
- **A** contains the remainder 2.
- **Q** contains the quotient 3

N	M	A	Q	Operation
4	00011	00000	1011	Initialize
	00011	00001	011_	Shift left AQ
	00011	11110	011_	A = A - M
	00011	00001	0110	Q[0] = 0 & restore A
3	00011	00010	110_	Shift left AQ
	00011	11111	110_	A = A - M
	00011	00010	1100	Q[0] = 0
2	00011	00101	100_	Shift left AQ
	00011	00010	100_	A = A - M
	00011	00010	1001	Q[0] = 1
1	00011	00101	001_	Shift left AQ
	00011	00010	001_	A = A - M
	00011	00010	0011	Q[0] = 1

Non-Restoring Division Algorithm

- More complex as compared to the restoring division algorithm.
- But when we implement this algorithm in hardware, it has an advantage, i.e., it contains only addition/subtraction per quotient bit.
- After subtraction, no restoring steps. Hence, the numbers of operations basically cut down up to half. Because of the less operation, the execution of this algorithm will be fast.

Flow Chart



- **Step 1:** The corresponding value will be initialized to the registers
  A ← 0, M ← Divisor, Q ← Dividend, and N ← number of bits in dividend.
- **Step 2:** Check the sign bit of A
- **Step 3:** If this bit of A is 1, then shift the value of AQ through left, and perform A = A + M. If this bit is 0, then shift the value of AQ into left and perform A = A - M. That means in case of 0, the 2's complement of M is added into register A, and the result is stored into A.
- **Step 4:** Check the sign bit of A again.
- **Step 5:** If this bit of register A is 1, then Q[0] will become 0. If this bit is 0, then Q[0] will become 1.
- **Step 6:** The value of N is decremented. Here N is used as a counter.
- **Step 7:** If the value of N = 0, then go to the next step. Otherwise, go to step 2.
- **Step 8:** A = A + M if the sign bit of register A is 1
- **Step 9:** This is the last step. Q contains quotient, and A contains remainder.

Example: 11 / 3
Dividend = 11
Divisor = 3

- If A = 0 Subtract else Add
- **A** contains the remainder 2
- **Q** contains the quotient 3

N	M	A	Q	Action
4	00011	00000	1011	Begin
	00011	00001	011_	Shift left AQ
	00011	11110	011_	A = A - M
3	00011	11110	0110	Q[0] = 0
	00011	11100	110_	Shift left AQ
	00011	11111	110_	A = A + M
2	00011	11111	1100	Q[0] = 0
	00011	11111	100_	Shift left AQ
	00011	00010	100_	A = A +M
1	00011	00010	1001	Q[0] = 1
	00011	00101	001_	Shift left AQ
	00011	00010	001_	A = A - M
0	00011	00010	0011	Q[0] = 1