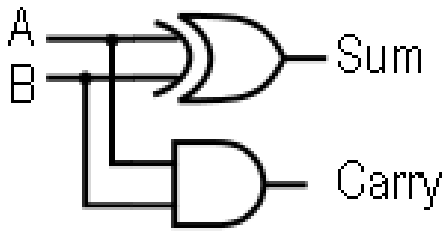
Review: Data representation in Digital systems

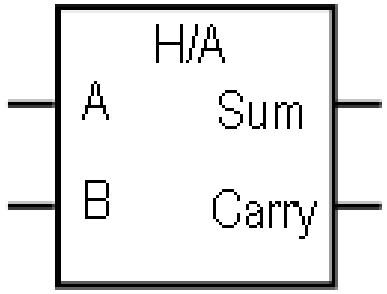
II: Operations in binary (and hex) system and related issues with hardware

Binary Addition (1): Tables and half adder

- To do binary additions we start with the usual tables. Since to write “two” we need two digits (10), the addition result is expressed with two bits:

A	B	"Carry"	"SUM"
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





Binary addition(2): Full addition table

Addition of two n-digit numbers requires the combination of three digit sums:

Example

Carries →		1	1	0			
Operand 1 →		1	0	1	1	+	11
Operand 2 →		0	1	1	0	=	6
Result →	1	0	0	0	1		17

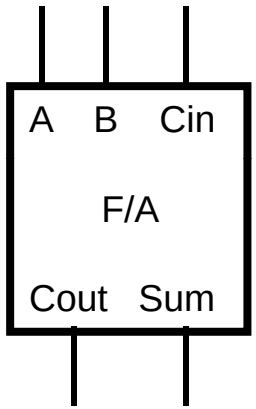
↑
Carry

Result may have (n+1) – digits;
Most significant one is called **CARRY**

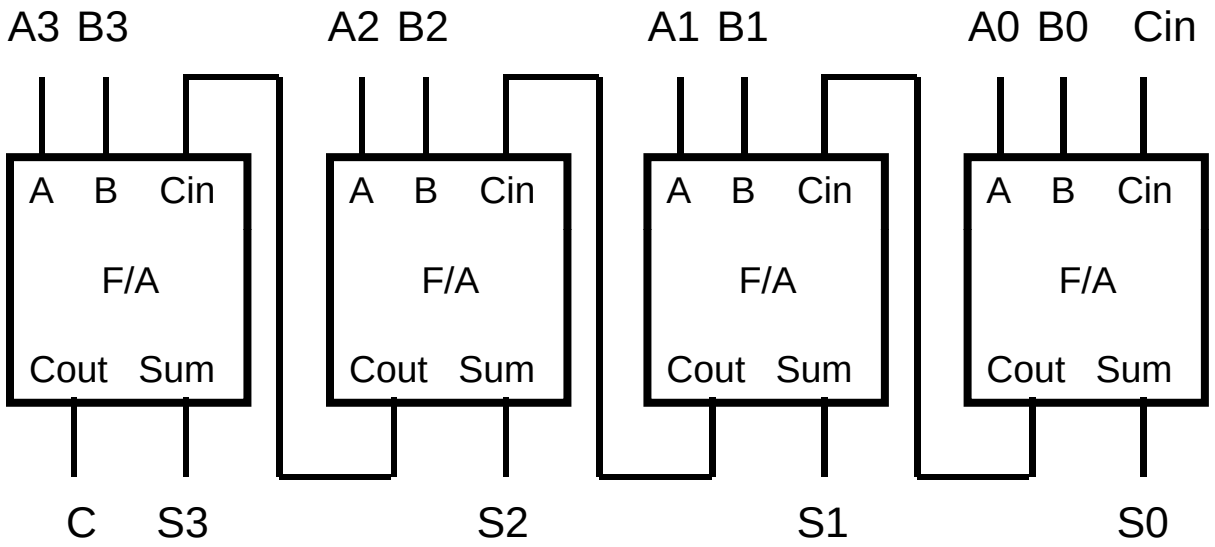
Full Addition Table

A	B	Cin	Cout	Sum
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

Binary addition (3): Full adder and n-bit adder



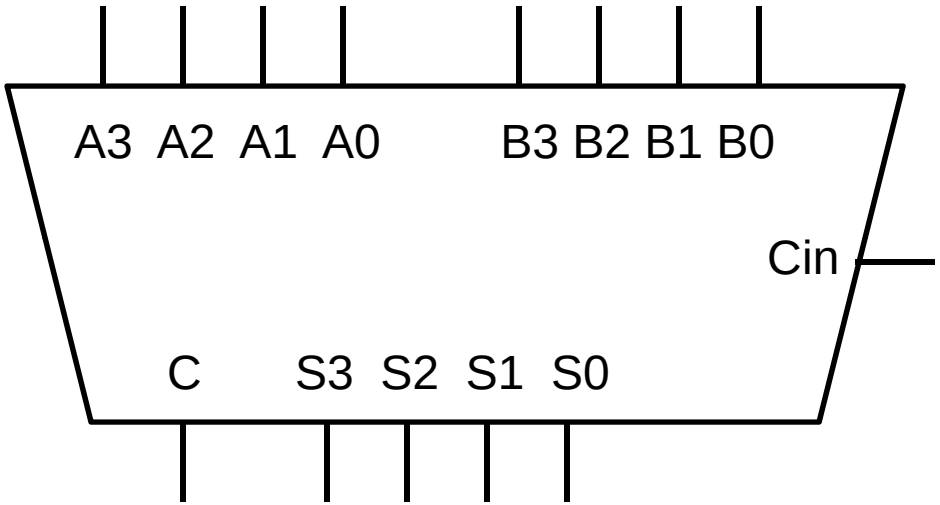
1-bit Full Adder

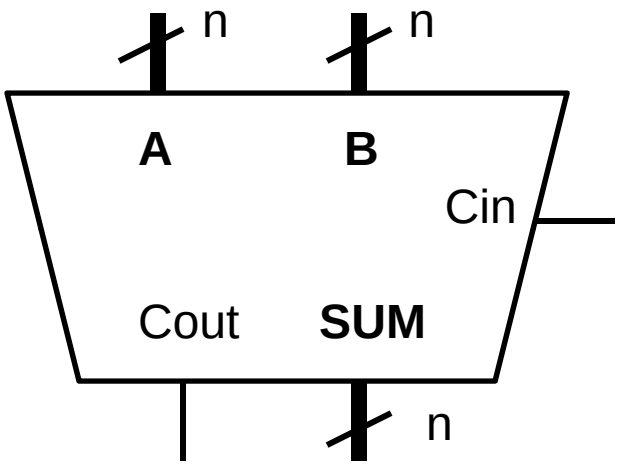


Example of an n-bit Full Adder

(n-bit adder)

Binary addition:
n-bit full adder considerations





1. The addition of two n-bit numbers generates in general an (n+1)-bit number
2. Two n-bit adders may be joined with the Cout-Cin pair to generate a 2n-bit adder
3. When (non negative) addition results must be n-bit long, C=1 indicates that we exceeded capabilities.
4. When C=1, we say that **there is a carry**.

Hex Addition

- Although hex addition tables are possible, we proceed by “extension”.
  - A carry occurs when we reach “sixteen”
  - The digit in the result when there is a carry is the excess over “sixteen”
  - Example:

```
   1  1  0
   2  A  3  E +    10814
   D  8  D  1 =    55505
-------------------------
1  0  3  0  F      66319
```

Binary Subtraction (1): Tables

- To do binary subtraction start with tables.
- If B > A, then A-B needs “a borrow”. Thus general case uses “two” bits

1 bit: A - B

A	B	“Borrow"	“Difference"
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Binary addition(2): Full Subtraction table

Subtraction with n-digit numbers requires the operating with three digits:

Example

Minuend	→		1	1	0	1	-	13
Subtrahend	→			1	1	1	-	7
Borrows	→		1	1	0		=	
Result	→	0	0	1	1	0		6

↑
Borrow

When Minuend > Subtrahend, Borrow = 0.

Full Subtraction Table: A-B-Borrow

A	B	Bin	Bout	Diff
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Subtraction: Considerations on Borrow

- For non negative numbers, A and B, the subtraction A-B with previous rules:
  - Generates a Borrow= 0 if $A \geq B$
  - Generates a Borrow = 1 if $A < B$

	1	1	0	1	-
	0	1	1	0	=
0	0	1	1	1	

Borrow

A > B

	0	1	1	0	-
	1	1	0	1	=
1	1	0	0	1	

Borrow

A < B

Subtraction by Complement's addition

- It is possible to build hardware substractors
- To reduce hardware complexity, subtraction may be realized by complement's addition.
- This reduces both operations to just one hardware adder plus extra gates

Principles of subtraction by complement addition

- In base r, r^n is written as 1 followed by n 0's
  - 5^2 = 100_5; 2^7=10000000B
- Assume P and Q have N digits (0 to left if necessary). Then
  - P – Q = P + (10^N – Q) – 10^N
- 10^N – Q is called **TEN's Complement**
  - In binary numbers, it is the **two's complement of Q**
  - If P≥Q, the addition produces a carry which is eliminated when subtracting 10^n
  - If P<Q, no carry is produced by addition and result is negative

Examples (In base 10) :

1000 – 087 = 913

198	-		198	+
087	=		913	=
111		**1**	**111**	-
		1	000	=
			111	

1000 – 198 = 802

087	-		087	+
198	=		802	=
-111		0	889	-
		1	000	=
			-111	

1. If the addition process generates a carry, then simply discard carry
2. If it does not generates a carry, result is **Negative of complement**
3. When subtraction is done following “borrow” rules,
    1. then borrow=0 if carry =1 in complement’s addition, and viceversa;
    2. Discarding borrow/carry, results are equal
4. This previous result is independent of the base.

198	-		198	+
087	=		913	=
0	**111**	**1**	**111**	-

087	-		087	+
198	=		802	=
1	889	0	889	-

Generating complements

- If X is the largest digit in system, then
  - 10^n-Q=(XXX…X – Q) + 1
  - Example: 10000h – 2F4Eh = FFFFh – 2F4E+1 = D0C1h+1 = D0C2h
- Binary CASE (X=1): Two's complement of b(n-1)b(n-2)…b1b0 is obtained complementing each bit and adding 1.

Subtraction with two's complement addition and hardware (1)

(1)

$$\begin{array}{r} A3\ A2\ A1\ A0\ - \\ \underline{B3\ B2\ B1\ B0\ =} \\ BW\ X3\ X2\ X1\ X0 \end{array} \quad \Longrightarrow \quad \begin{array}{r} 1\ + \\ A3\ A2\ A1\ A0\ + \\ \underline{\overline{B3}\ \overline{B2}\ \overline{B1}\ \overline{B0}\ =} \\ CY\ X3\ X2\ X1\ X0 \end{array}$$

with $CY = \overline{BW}$

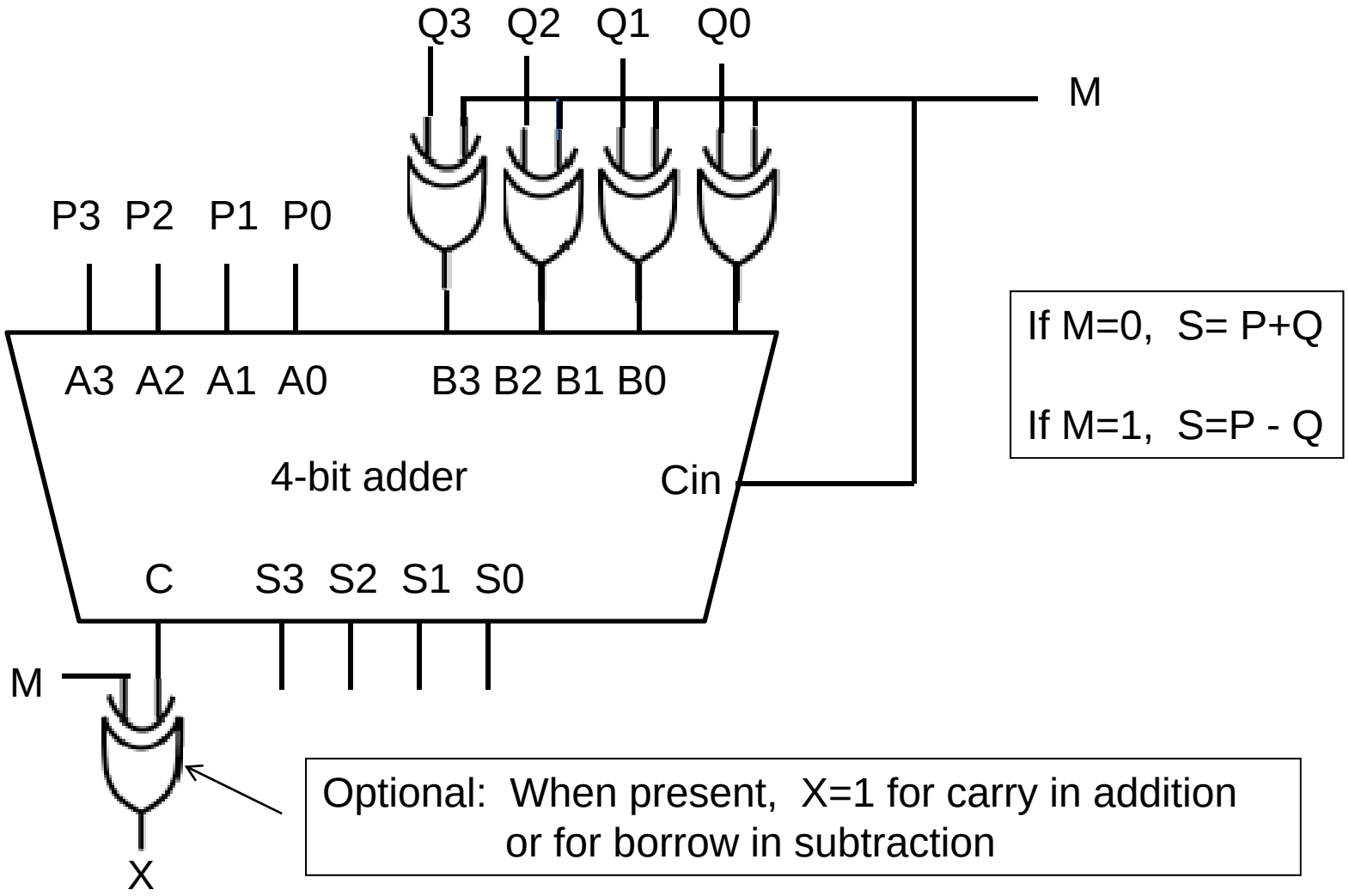
(2)

XOR PROPERTY:

$$0 \oplus X = X$$

$$1 \oplus X = \overline{X}$$

Subtraction with two's complement addition and hardware (2)



Multiplication and Division

1. Hand Business as usual

```
      1 1 0  x
        1 1  =
    ---------
      1 1 0
    1 1 0
  ---------
  1 0 0 1 0
```

```
          1 0 0
      ---------
1 1 | 1 1 0 1
      1 1
    -------
        0 0 1
```

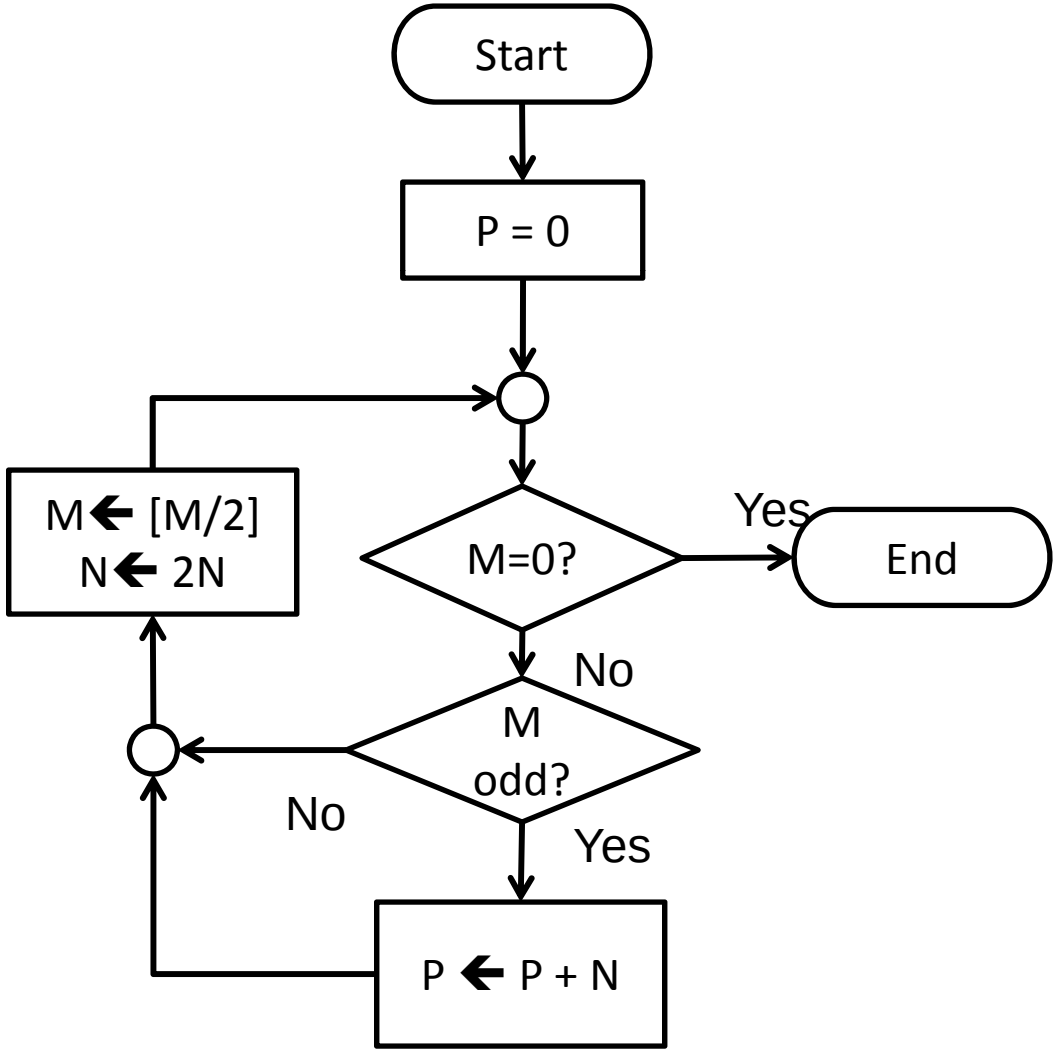
2. Hardware multipliers/dividers exist but may be complicated.
3. In digital systems, multiplication and division may be realized by software
4. Inefficient but sure software methods:
   - 4.1 M x N: add M N times
   - 4.2 M/N: Subtract N until result is less than N; result is residue and the number of subtractions is the quotient.
5. Better methods: left shift-and-add for multiplication and right-shift and subtract for division (search in literature or internet)

Multiplication: The peasant's algorithm: P = MxN (integers)

A popular software algorithm for integer multiplication:

1. Initialize product P=0
2. WHILE M>0
   - 2.1 If M is odd, P=P+N
   - 2.2 M = M/2 (integer div)
   - 2.3 N = 2N

   END WHILE



Multiplication:
Example 163x215 = 35 045

Running algorithm:

Step	M	N	P
	163	215	
1	163	215	0
2.1	163	215	215
2.2 & 2.3	81	430	215
2.1	81	430	645
2.2 & 2.3	40	860	645
2.1	40	860	645
2.2 & 2.3	20	1720	645
2.1	20	1720	645
2.2 & 2.3	10	3440	645
2.1	10	3440	645
2.2 & 2.3	5	6880	645
2.1	5	6880	7525
2.2 & 2.3	2	13760	7525
2.1	2	13760	7525
2.2 & 2.3	1	27520	7525
2.1	1	27520	35045
2.2 & 2.3	0	55040	35045

STOP

Normal "hand" process:

M	N	To add
163	215	215
81	430	430
40	~~860~~	0
20	~~1720~~	0
10	~~3440~~	0
5	6880	6880
2	~~13760~~	0
1	27520	27520
0	~~55040~~	
P =		**35045**

Proof can be developed by using binary expression for M

Data representation in Digital systems

IV: Digital representation of Numbers

Facts to Consider

- Only bits can be used: 0 and 1
- Only a finite number of bits can be used:
  - n bits yield only 2^n different numbers
  - It is always possible to say which is the following number
- Valid operations may yield invalid results
  - Overflow (and underflow)

Representations to consider:

Used in this course:

NUMBER

INTEGERS

REAL
(integer and fractional)

Unsigned

Signed

Fixed Potin

(Signed and unsigned)

1. Normal binary
2. BCD
3. Biased

1. Two's complement
2. Biased

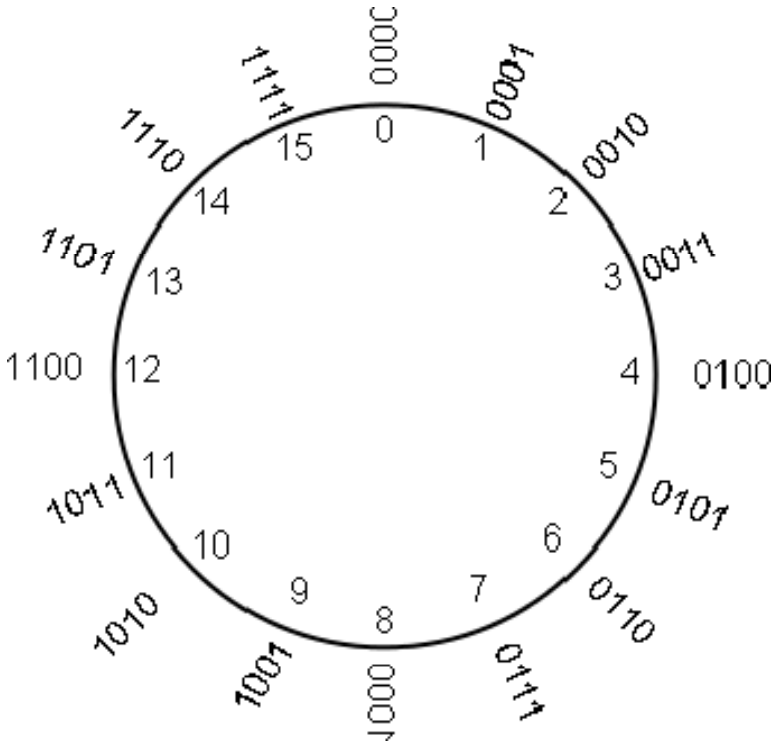
In this course we talk of

- Integer numbers
  - Unsigned integers: normal binary, Biased, BCD
  - Signed Integers: Two's complement, biased
- Real Numbers
  - Fixed point format: signed and unsigned
- Continuous intervals
  - Regular subintervals and ½ -LSB offset intervals
- Non numerical data
  - ASCII, Unicode
  - Custom

Unsigned normal binary (1)

- With N digits, interval 0 to 2^N -1
- Examples:
  - 3 digits: [0, 7]
  - Nibbles: [0, 15]
  - Bytes: [0, 255]
  - Words: [0, 65 535]
  - Double words: [0, 4 294 967 295]

Unsigned Normal binary (2): Graphical representation



‘ 1 1 1 0 +
‘ 0 1 1 0 =
‘1 **0 1 0 0**

14 + 6 = 4
Carry shows overflow

‘ 0 1 1 1 +
‘ 1 1 0 1 =
‘1 **1 0 1 0**

7 – 13 = 10

Borrow shows underflow (overflow)