Chapter 4.

Number Theory and Cryptography

Part I: The Integers and Division

Dept. of Software
Gachon University
Spring 2022

Contents

- Divisibility and Modular Arithmetic
- Integer Representations and Algorithms

4.1 Divisibility and Modular Arithmetic

Introduction

- Of course you already know what the integers are, and what division is…

- **But:** There are some specific notations, terminology, and theorems associated with these concepts which you *may not* know.

- These form the basics of *number theory*.
  - Vital in many important algorithms today (*hash functions*, *cryptography*, *digital signatures*).

Division; Factor and Multiple

- Let a, $b \in \mathbf{Z}$ with $a \neq 0$.
- $a|b \;\equiv$ "*a divides b*" $:\equiv$ "$\exists c \in \mathbf{Z}$: $b=ac$"
  "There is an integer *c* such that *c* times *a* equals *b*."
- Otherwise $a \nmid b$
  - Example: $3|12 \Leftrightarrow$ **True**, but $3|7 \Leftrightarrow$ **False**. $3 \nmid 7$
- If *a* divides *b*, then we say *a* is a *factor* or a *divisor* of *b*, and *b* is a *multiple* of *a*.

- "*b* is even" $:\equiv 2|b$.

Division : Properties of Divisibility

- $\forall a,b,c \in \mathbf{Z}$:
  1. $a|0$ $(2|0, 3|0, \ldots)$
  2. $(a|b \wedge a|c) \rightarrow a \mid (b+c)$ $(2|4 \wedge 2|6 \rightarrow 2|10)$
  3. $a|b \rightarrow a|bc$ $(2|4 \rightarrow 2|4 \cdot 3)$
  4. $(a|b \wedge b|c) \rightarrow a|c$ $(2|4 \wedge 4|8 \rightarrow 2|8)$

- **Proof** of (2) : next page

Cont.

- Show $\forall a,b,c \in \mathbf{Z}: (a|b \wedge a|c) \rightarrow a \mid (b + c)$.

Division "Algorithm"

※ Really just a *theorem*, not an algorithm…

- $\forall a,d \in \mathbf{Z}, d>0: \exists! q,r \in \mathbf{Z}: 0 \le r < |d|, a=dq+r.$ ($\exists$! means "unique")
- Let a be an integer and d a positive integer
- Then there are unique integers q and r such that $a=dq+r$ where $0 \le r < d$

  - d is the divisor ("제수")
  - a is the dividend ("피제수")
  - q is the quotient ("몫")
  - r is the remainder ("나머지")

- q = a **div** d
- r = a **mod** d

  – We can find q and r by: $q=\lfloor a/d \rfloor$, $r = a - qd$.

  (e.g., if a = 14 and d = 3, then $q = \lfloor 14/3 \rfloor = 4$ and $r = 14 - 3 \cdot 4 = 2$.

Question

- In C programming, how to implement the following condition ?
  - Write a C program that reads **an integer N** and do the following:
    - If N is positive, print “positive integer”
    - If N is positive and **even**, print “even integer”
    - Otherwise “integer”

Modular Arithmetic : mod Operator

- An integer "division remainder" operator.
- Let $a,d \in \mathbf{Z}$ with $d>1$, then
  - $\boxed{a \textbf{ mod } d}$ denotes the remainder r, i.e., the remainder when a is divided by d.
    - $r = a \textbf{ mod } d$
- We can compute (a **mod** d) by: $a - d \cdot \underbrace{\lfloor a/d \rfloor}_{q}$.
- In C programming language, "%" = mod.

Modular Arithmetic: Congruence Relation $a \equiv b \pmod{m}$

- Let $\mathbf{Z}^+=\{n\in\mathbf{Z} \mid n>0\}$, the positive integers. Let $a,b\in\mathbf{Z}$, $m\in\mathbf{Z}^+$.

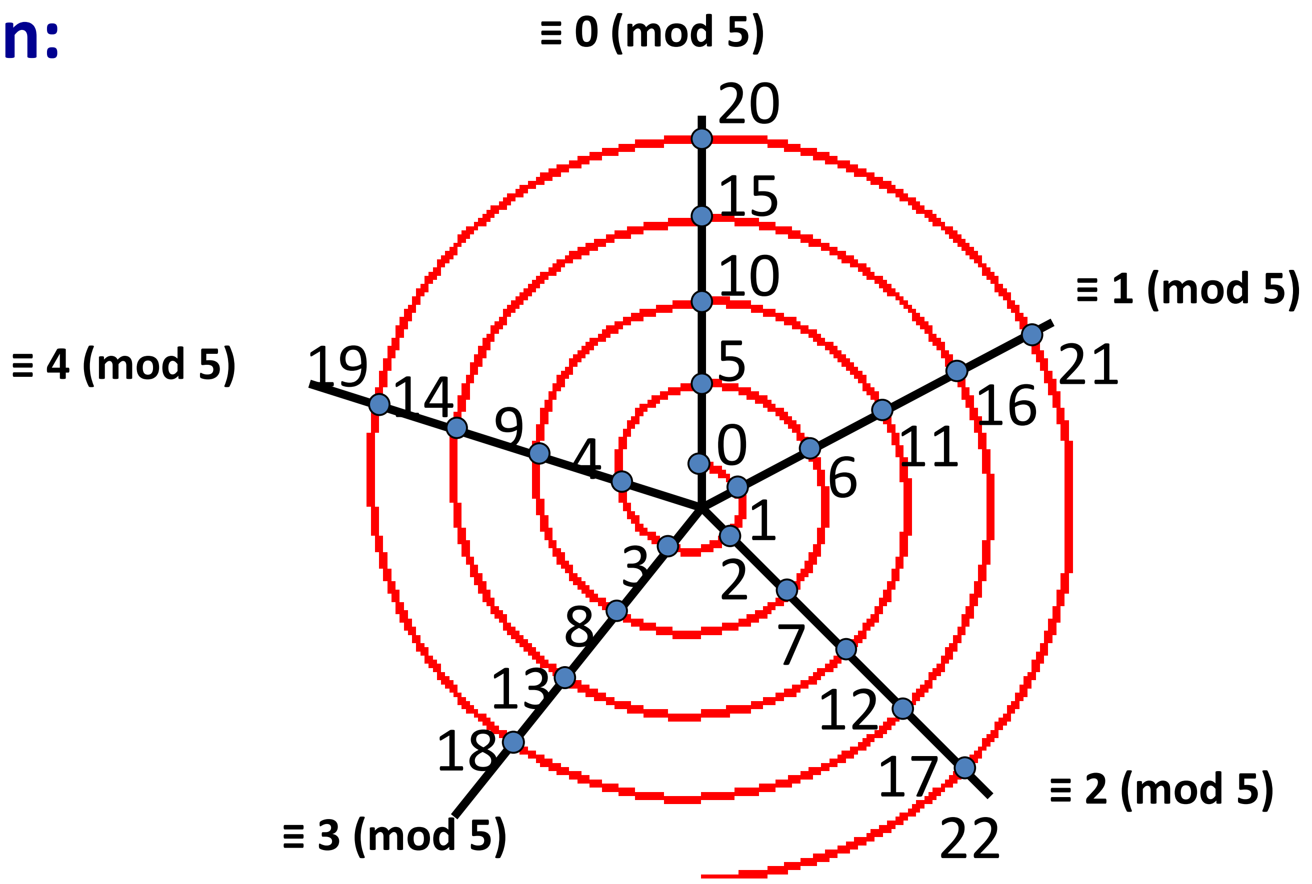
DEFINITION:

- $a \equiv b \pmod{m}$
  - *a* is congruent to *b* modulo *m* *iff* $m \mid a - b$.

- Also equivalent to: $(a-b) \bmod m = 0$.
- Example
  - $17 \equiv 5 \pmod{6}$
  - $24 \not\equiv 14 \pmod{6}$
- Example problem :
  - What time it will be (on a 24-hour clock) 50 hours from now ?

Spiral Visualization of **mod**

- **Example shown: modulo-5 arithmetic**



Modular Arithmetic – Useful Theorems

- (Theorem 4) Let $a,b \in \mathbf{Z}$, $m \in \mathbf{Z}+$. Then:

  $$a \equiv b \ (\text{mod } m) \Leftrightarrow \exists k \in \mathbf{Z}\ a = b + km.$$

- (Theorem 5) Let $a,b,c,d \in \mathbf{Z}$, $m \in \mathbf{Z}+$. If $a \equiv b \ (\text{mod } m)$ and $c \equiv d \ (\text{mod } m)$, then:

  $a + c \equiv b + d \ (\text{mod } m)$, and
  $ac \equiv bd \ (\text{mod } m)$

Applications of Modular Arithmetic (참조만)

- The **mod** operator is widely used in *hash functions*.
  - $h(key) = key \bmod m$
- *Linear congruential methods* is used to generate *pseudo random numbers*.
  - $x[n+1] = (a \cdot x[n] + c) \bmod m$
- Also, in cryptography, encryption, …

4.2 Integer Representations and Algorithms

Representations of Integers

- In the modern world, we use *decimal,* or *base* 10, *notation* to represent integers. For example when we write 965, we mean $9 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0$.

- We can represent numbers using any base *b*, where *b* is a positive integer greater than 1.

Representations of Integers

- Base-*b* representations of integers.
  - Especially: **binary (b=2), hexadecimal (b=16) , octal (b=8).**
  - Also, two's complement representation
- Algorithms for computer arithmetic:
  - Binary addition, multiplication, division.
- Euclidean algorithm for finding GCD's.

Base-b Representations of Integers

- If b is a positive integer greater than 1, then a given positive integer $\boldsymbol{n}$ can be uniquely represented as follows:

  $$\boldsymbol{n} = a_k b^k + a_{k-1} b^{k-1} + \ldots + a_1 b^1 + a_0 b^0$$

  where
  - k is a natural number.
  - and a_0, a_1, …, and a_k are a natural number less than b.
  - $a_k \neq 0$.

- Example:
  - $165 = 1 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0 = (165)_{10}$
  - $165 = 2 \cdot 8^2 + 4 \cdot 8^1 + 5 \cdot 8^0 = (245)_8$

Base-b Number Systems

- Ordinarily we write *base*-10 representations of numbers (using digits 0-9).
- However, 10 isn't special; any base b>1 will work.
- For any positive integers n, b, there is a unique sequence

$$\underbrace{a_k a_{k-1} \ldots a_1 a_0}_{\text{The "base } b \text{ expansion of } n\text{"}} \text{ of } \textit{digits } a_i < b \text{ such that}$$

Particular Bases of Interest

- Base *b*=10 (decimal):

  10 digits: 0,1,2,3,4,5,6,7,8,9.

- Base *b*=2 (binary):

  2 digits: 0,1. ("Bits"="binary digits.")

- Base *b*=8 (octal):

  8 digits: 0,1,2,3,4,5,6,7.

- Base *b*=16 (hexadecimal):

  16 digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Used only because we have 10 fingers

Used internally in all modern computers

Octal digits correspond to groups of 3 bits

Hex digits give groups of 4 bits

Binary Expansions

- Most computers represent integers and do arithmetic with binary (base 2) expansions of integers. In these expansions, the only digits used are 0 and 1.

- **Example**: What is the decimal expansion of the integer that has $(1\ 0101\ 1111)_2$ as its binary expansion?
  - **Solution**:
  - $(1\ 0101\ 1111)_2 = 1\cdot2^8 + 0\cdot2^7 + 1\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$ =351.
- **Example**: What is the decimal expansion of the integer that has $(11011)_2$ as its binary expansion?
  - **Solution**:
  - $(11011)_2 = 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$ =27.

Converting to Base b (1/2)

- Informal Algorithm (the base b expansion of an integer n)

1. To convert any integer n to any base b (b>1):
2. To find the value of the *rightmost* (lowest-order) digit, simply compute n mod b.
3. Now replace n with the quotient $\lfloor n/b \rfloor$.
4. Repeat above two steps to find subsequent digits, until n is gone (=0).

- $(177130)_{10} = (?)_{16}$
  - $177130 = 16 \cdot 11070 + \mathbf{10}$
  - $11070 = 16 \cdot 691 + \mathbf{14}$
  - $691 = 16 \cdot 43 + \mathbf{3}$
  - $43 = 16 \cdot 2 + \mathbf{11}$
  - $2 = 16 \cdot 0 + \mathbf{2}$
  - ➔ $(177130)_{10} = (2B3EA)_{16}$

- $(241)_{10} = (?)_2$
  - $241 = 2 \cdot 120 + \mathbf{1}$, $120 = 2 \cdot 60 + \mathbf{0}$
  - $60 = 2 \cdot 30 + \mathbf{0}$, $30 = 2 \cdot 15 + \mathbf{0}$
  - $15 = 2 \cdot 7 + \mathbf{1}$, $7 = 2 \cdot 3 + \mathbf{1}$
  - $3 = 2 \cdot 1 + \mathbf{1}$, $1 = 2 \cdot 0 + \mathbf{1}$
  - ➔ $(241)_{10} = (11110001)_2$

Converting to Base b (2/2)

- Formal Algorithm

procedure *base b expansion* (n: positive integer)

$q := n$

$k := 0$

while $q \neq 0$

begin

$a_k := q$ **mod** b {remainder}

$q := \lfloor q/b \rfloor$ {quotient}

$k := k + 1$

end {the base b expansion of n is $(a_k a_{k-1} \ldots a_1 a_0)_b$}

- q represents the quotient obtained by successive divisions by b, starting with $q = n$.
- The digits in the base b expansion are the remainders of the division given by q **mod** b.
- The algorithm terminates when $q = 0$ is reached.

Exercise

- **Example**: Find the octal expansion of $(12345)_{10}$
-

Conversion Between Binary, Octal, and Hexadecimal Expansions

- **Example**: Find the octal and hexadecimal expansions of $(11\ 1110\ 1011\ 1100)_2$.

- **Solution**:
  - To convert to octal, we group the digits into blocks of three $(011\ 111\ 010\ 111\ 100)_2$, adding initial 0s as needed. The blocks from left to right correspond to the digits $3,7,2,7$, and 4. Hence, the solution is $(37274)_8$.
  - To convert to hexadecimal, we group the digits into blocks of four $(0011\ 1110\ 1011\ 1100)_2$, adding initial 0s as needed. The blocks from left to right correspond to the digits $3,\mathrm{E},\mathrm{B}$, and C. Hence, the solution is $(3\mathrm{EBC})_{16}$.

Cont.

TABLE 1 Hexadecimal, Octal, and Binary Representation of the Integers 0 through 15.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Octal	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

- Each octal digit corresponds to a block of 3 binary digits.
- Each hexadecimal digit corresponds to a block of 4 binary digits.
- So, conversion between binary, octal, and hexadecimal is easy.

Binary to Octal or Hexadecimal

- **Binary to Octal**

Binary: 11100101 = 011 100 101

Binary:	000	001	010	011	100	101	110	111
Octal:	0	1	2	3	4	5	6	7

Binary = 011 100 101
Octal = 3 4 5

- **Binary to Hexadecimal**

Binary: 11100101 = 1110 0101

Binary:	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal:	0	1	2	3	4	5	6	7
Binary:	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal:	8	9	A	B	C	D	E	F

Binary = 1110 0101
Hexadecimal = E 5

Addition of Binary Numbers

- Intuition (let $a = (a_{n-1} \ldots a_1 a_0)_2$, $b = (b_{n-1} \ldots b_1 b_0)_2$)

$$
\begin{array}{lclllll}
 & & & c_{n-1} & c_{n-2} \ldots & c_1 & c_0 \\
a & = & & a_{n-1} & a_{n-2} \ldots a_2 & a_1 & a_0 \\
b & = & & b_{n-1} & b_{n-2} \ldots b_2 & b_1 & b_0 \\
\hline
a+b & = & s_n & s_{n-1} & s_{n-2} \ldots s_2 & s_1 & s_0
\end{array}
$$

$c_i = \lfloor (a_{i-1}+b_{i-1}+c_{i-1})/2 \rfloor$

$s_i = (a_i+b_i+c_i)\%2$

- Algorithm

procedure *add*($a_{n-1} \ldots a_0$, $b_{n-1} \ldots b_0$: binary expressions of a,b)
  $c := 0$ {c mean a carry}
  for $i := 0$ **to** $n-1$ {i means a bit index}
  begin
    $sum := a_i + b_i + c$ {2-bit sum}
    $s_i := sum$ **mod** 2 {low bit of sum}
    $c := \lfloor sum/2 \rfloor$ {high bit of sum}
  end
  $s_n := c$
  {the binary expression of the sum is $(s_n s_{n-1} \ldots s_1 s_0)_2$}

O(n)

- $\boldsymbol{n} = a_k 2^k + a_{k-1} 2^{k-1} + \ldots + a_1 2^1 + a_0 2^0$

- $n_1 := a_k = 1,\ \ a_{k-1} = a_{k-2} \ldots = a_1 = 0$

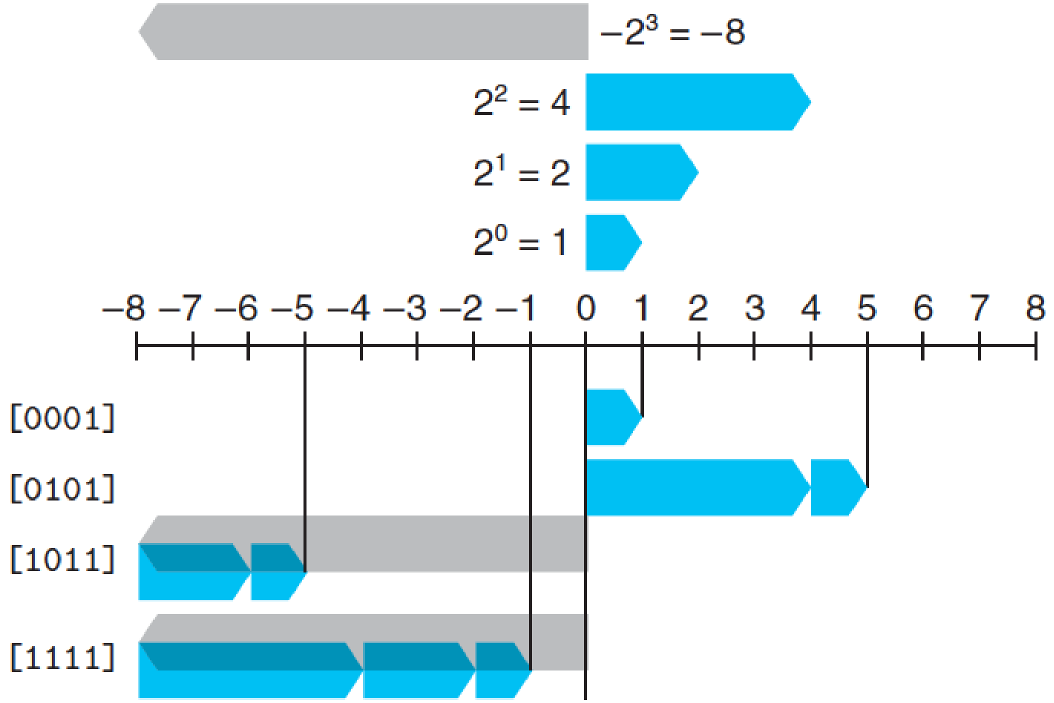
  vs.
- $n_2 := a_k = 0,\ \ a_{k-1} = a_{k-2} \ldots = a_1 = 1$

Unsigned vs. Signed numbers

- Two's-Complement Encodings

$$B2T_w(\vec{x}) \doteq -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

$$\begin{array}{lllllll}
B2T_4([0001]) & = & -0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 & = & 0+0+0+1 & = & 1 \\
B2T_4([0101]) & = & -0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 & = & 0+4+0+1 & = & 5 \\
B2T_4([1011]) & = & -1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 & = & -8+0+2+1 & = & -5 \\
B2T_4([1111]) & = & -1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 & = & -8+4+2+1 & = & -1
\end{array}$$



2's Complement (1/2)

- In binary, negative numbers can be conveniently represented using **2**'*s complement notation*.
- In this scheme, **a string of *n* bits** can represent integers **$-2^{n-1} \sim (2^{n-1}-1)$.**
  - Unsigned integer … $0 \sim 2^n-1$ (e.g., `unsigned int` n)
  - Integer … $-2^{n-1} \sim 2^{n-1}-1$ (e.g., `int` n)
- The bit in the highest-order bit-position ($n-1$) (leftmost bit) represents a coefficient multiplying -2^{n-1};
  – The other positions $i < n-1$ just represent 2^i, as before.

2's Complement (2/2)

- The negation of any n-bit 2's complement number $a(= a_{n-1}\ldots a_0)$ is given by $\boxed{\overline{a_{n-1}\ldots a_0}} + 1$.

  Bitwise logical complement of ***a***

- Examples
  - $1011 = -(0100 + 1) = -(0101) = -(5)_{10}$
  - $0100 = +0100 = (4)_{10}$

Subtraction of Binary Numbers

- **Theorem**: For an integer a represented in 2's complement notation, $-a = \bar{a} + 1$.

  Proof: Just try it by yourself!

- **Algorithm**

procedure *subtract* $(a_{n-1}\ldots a_0,\ b_{n-1}\ldots b_0$:
  binary 2's complement expressions of $a,b)$
  return $add(a,\ add(\bar{b},\ 1))$ $\{\ a + (-b)\ \}$

Binary Multiplication of Integers (1/2)

- Intuition (let $a = (a_{n-1} \dots a_1 a_0)_2$, $b = (b_{n-1} \dots b_1 b_0)_2$)

$$
\begin{array}{llcccccccccc}
a & = & & & & a_{n-1} & a_{n-2} & \cdots & a_2 & a_1 & a_0 \\
b & = & & & & b_{n-1} & b_{n-2} & \cdots & b_2 & b_1 & b_0 \\
\hline
c_0 & = & & & & s_{(n-1,0)} & s_{(n-2,0)} & \cdots & s_{(2,0)} & s_{(1,0)} & s_{(0,0)} \\
c_1 & = & & & s_{(n-1,1)} & s_{(n-2,1)} & \cdots & s_{(2,1)} & s_{(1,1)} & s_{(0,1)} & 0 \\
c_2 & = & & s_{(n-1,2)} & s_{(n-2,2)} & \cdots & s_{(2,2)} & s_{(1,2)} & s_{(0,2)} & 0 & 0 \\
 & & & \cdots\cdots & & & & & & & \\
+) & & & & & & & & & & \\
\hline
a \cdot b & = & \multicolumn{9}{l}{c_{n-1} + c_{n-2} + \dots + c_2 + c_1 + c_0}
\end{array}
$$

$s_{(i,j)}$ = (if $b_j = 1$ then a_i else 0)
c_j = (if $b_j = 1$ then $a \ll j$ else 0)

Binary Multiplication of Integers (2/2)

ALGORITHM 3 Multiplication of Integers.

procedure *multiply*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1}a_{n-2}\ldots a_1a_0)_2$
  and $(b_{n-1}b_{n-2}\ldots b_1b_0)_2$, respectively}
for $j := 0$ **to** $n - 1$
  if $b_j = 1$ **then** $c_j := a$ shifted j places
  else $c_j := 0$
{$c_0, c_1, \ldots, c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ **to** $n - 1$
  $p := p + c_j$
return p {p is the value of ab}

- Ex 10. Find the product of $a = (110)_2$ and $b = (101)_2$.

Division Algorithm

- Example: 23/4?

		r	q
23 – 4	=	19	1
19 – 4	=	15	2
15 – 4	=	11	3
11 – 4	=	7	4
7 – 4	=	**3**	**5**

q: the number of times we perform this subtraction

- Algorithm

ALGORITHM 4 Computing div and mod.

procedure *division algorithm*(a: integer, d: positive integer)
$q := 0$
$r := |a|$
while $r \geq d$
  $r := r - d$
  $q := q + 1$
if $a < 0$ and $r > 0$ **then**
  $r := d - r$
  $q := -(q + 1)$
return (q, r) $\{q = a$ **div** d is the quotient, $r = a$ **mod** d is the remainder$\}$

Section Summary

- Integer Representations
  - Base *b* Expansions
  - Binary Expansions
  - Octal Expansions
  - Hexadecimal Expansions
- Base Conversion Algorithm
- Algorithms for Integer Operations