東北大学 工学部 機械知能・航空工学科
2020年度 クラス C D

情報科学基礎 I

2. 数の表現 — 符号なし整数

(教科書1.1節)

大学院情報科学研究科
鏡 慎吾

2進数の基礎

r 進数

10 進数:

0, 1, 2, … 9 の 10 個のシンボルを使って数を表す

$$1234_{(10)} = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

同様に,

r 個のシンボルを使って表したものを r 進数と呼ぶ

$$1234_{(r)} = 1 \times r^3 + 2 \times r^2 + 3 \times r^1 + 4 \times r^0$$

シンボルの数 r を基数 (radix) と呼ぶ

2進数，8進数，16進数

10進数	2進数	8進数	16進数
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	a
11	1011	13	b
12	1100	14	c
13	1101	15	d
14	1110	16	e
15	1111	17	f
16	10000	20	10
17	10001	21	11
18	10010	22	12

10進数との変換

2進から10進: 定義どおりに計算する

$$1001_{(2)}$$
$$= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 9_{(10)}$$

10進から2進: 2のべき乗の和に分解する

[方法1] 下位ビットから決めていく

2) 37
2) 18 … 1
2) 9 … 0
2) 4 … 1
2) 2 … 0
2) 1 … 0
0 … 1 → $100101_{(2)}$

[方法2] 上位ビットから決めていく

37 = 32 + 5
= 32 + 4 + 1
= $100101_{(2)}$

方法2は，2のべき乗数を暗記していないとできない．しかし慣れるとこの方が早い

加算・減算

加算: 10進では，各桁の加算結果が9を超えると繰り上がる
　　2進では，1を超えると繰り上がる

$$
\begin{array}{rll}
 & 1111 & \\
 & 01111010_{(2)} & = 122_{(10)} \\
+) & 00111001_{(2)} & = 57_{(10)} \\
\hline
 & 10110011_{(2)} & = 179_{(10)}
\end{array}
$$

減算: 10進では，繰り下がりで10を借りてくる.
　　2進では，2を借りてくる

$$
\begin{array}{rll}
 & 0 & \\
 & 01111010_{(2)} & = 122_{(10)} \\
-) & 00111001_{(2)} & = 57_{(10)} \\
\hline
 & 01000001_{(2)} & = 65_{(10)}
\end{array}
$$

乗算・除算

加算・減算が理解できれば，基本的には同じである

- 九九は不要; AND だけが分かればよい

$$
\begin{array}{rl}
 & 0111_{(2)} = 7_{(10)} \\
\times) & 1101_{(2)} = 13_{(10)} \\
\hline
 & 0111 \\
 & 0000 \\
 & 0111 \\
 & 0111 \\
\hline
 & 1011011_{(2)} = 91_{(10)}
\end{array}
$$

$$
\begin{array}{rl}
 & 1011_{(2)} = 11_{(10)} \\
\hline
1011_{(2)}) & 01111010_{(2)} = 122_{(10)}/11_{(10)} \\
 & 1011 \\
\hline
 & 10001 \\
 & 1011 \\
\hline
 & 1100 \\
 & 1011 \\
\hline
 & 1_{(2)} = \ldots 1_{(10)}
\end{array}
$$

シフト演算

10010 **001101**01　を右2ビットシフトすると，
100 **10001101**　になる

11001010 **11111110**　を左3ビットシフトすると，
11001010111 **11110**000　になる

- 10進数: 左 k 桁シフトは 10^k 倍，右 k 桁シフトは 10^{-k} 倍
- 2進数: 左 k ビットシフトは 2^k 倍，右 k ビットシフトは 2^{-k} 倍
  (ただし端数は無視)

C言語では

```
17  >>  3
 9  <<  4
```

（右3ビットシフト）
（左4ビットシフト）
のように書く

8進数，16進数との変換

8進数への変換: 3桁ごとに区切ってパターンを置き換える

001000110101011

1 0 6 5 3

(C言語では `010653`)

16進数への変換: 4桁ごとに区切ってパターンを置き換える

1001000110101011

9 1 a b

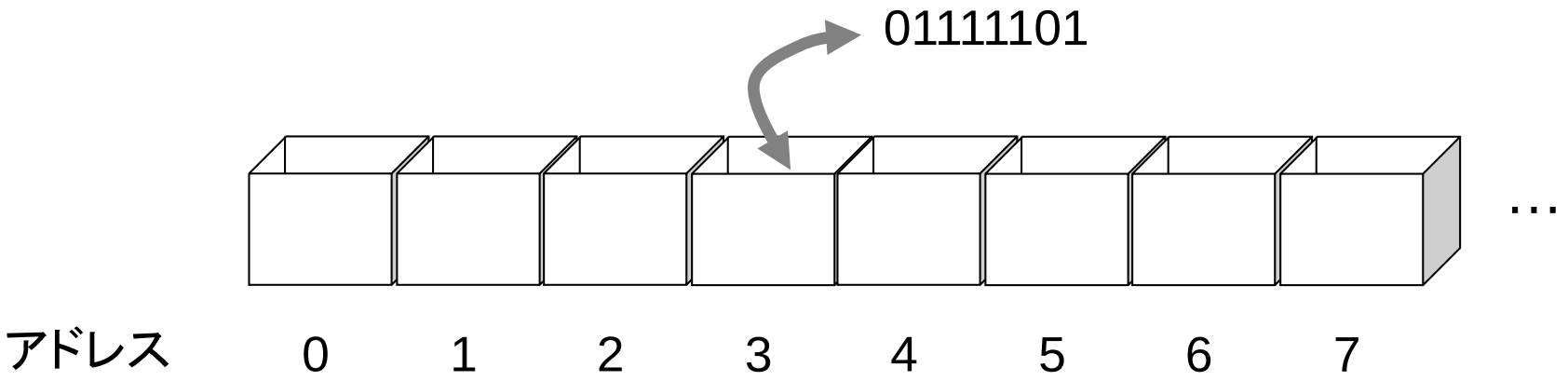
(C言語では `0x91ab`)

2進数への変換も，これらの逆の操作をするだけ

桁数が有限の場合

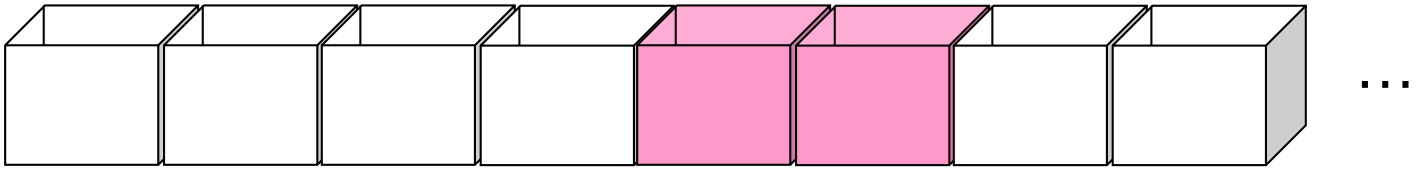
ビットとバイト

- 2進数の1桁を bit (binary digit) と呼ぶ
- 計算機の中では，有限の桁数でデータを表現しなくてはならない
- 基本単位: 1 byte = 8 bit
- メモリは 1 バイトごとに区切られて，それぞれにアドレスがついている



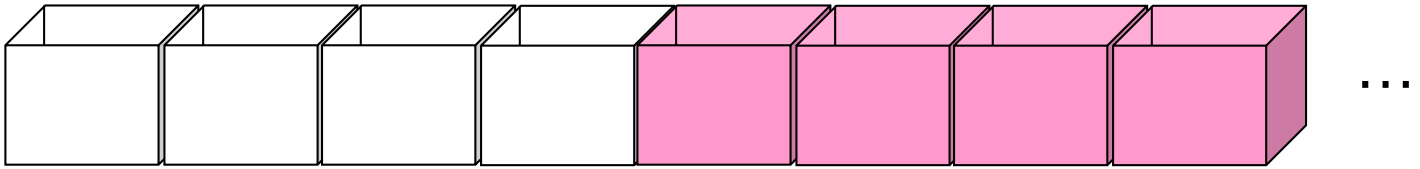
複数のバイトをまとめて扱う

メモリアドレス 4 に対する 2 バイト数値のアクセス



アドレス　0　1　2　3　4　5　6　7

メモリアドレス 4 に対する 4 バイト数値のアクセス



アドレス　0　1　2　3　4　5　6　7

複数ビットの呼び方

	ほとんどの32ビットプロセッサ（と互換性を保つ64ビットプロセッサ）	ほとんどの16ビットプロセッサ（と互換性を保つ32ビットプロセッサ）	通信分野	C の型名（最近のほとんどの計算機の場合）
8 bit	byte	byte	octet	`char`
16 bit	half word	word		`short int`
32 bit	word	double word		`int`
64 bit	double word			`long long int`

- 一度に処理できるサイズによって，16ビットプロセッサ，32ビットプロセッサ，などと分類する
- word の定義はプロセッサによって違う（過去には byte の定義が違うプロセッサも存在した）
- C言語の型のサイズすら，計算機によって違う
  - 最近のコンパイラでは `int32_t` などの型名も使える

表せる数値の範囲（符号なしの場合）

- 一般に，n ビット長のデータは 2^n 種類の値を表現できる
- nビット符号なし数: 0 ～ $2^n - 1$ の整数を表す

  8ビット符号なし数: 0 ～ 255

  16ビット符号なし数: 0 ～ 65,535

  32ビット符号なし数: 0 ～ 4,294,967,295
- C言語では，整数型名の前に `unsigned` をつけると符号なし数になる

  `unsigned int, unsigned short, …`

- $2^{10} = 1024$ なので，2^{10} 倍を kilo で表す場合がある．この分野では，1k が 1024 なのか 1000 なのか注意が必要．
  - 同様に，2^{20} 倍 = mega，2^{30} 倍 = giga
- 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 程度までは覚えておくと便利
  - 「2^{24} はだいたい幾つか?」$2^{24} = 2^{10} \times 2^{10} \times 2^4$ だから，100万の 16倍くらい，とすぐに見当がつく

MSB と LSB

現在考えている桁数で最上位のビットと最下位のビットをそれぞれ MSB, LSBと呼ぶ.

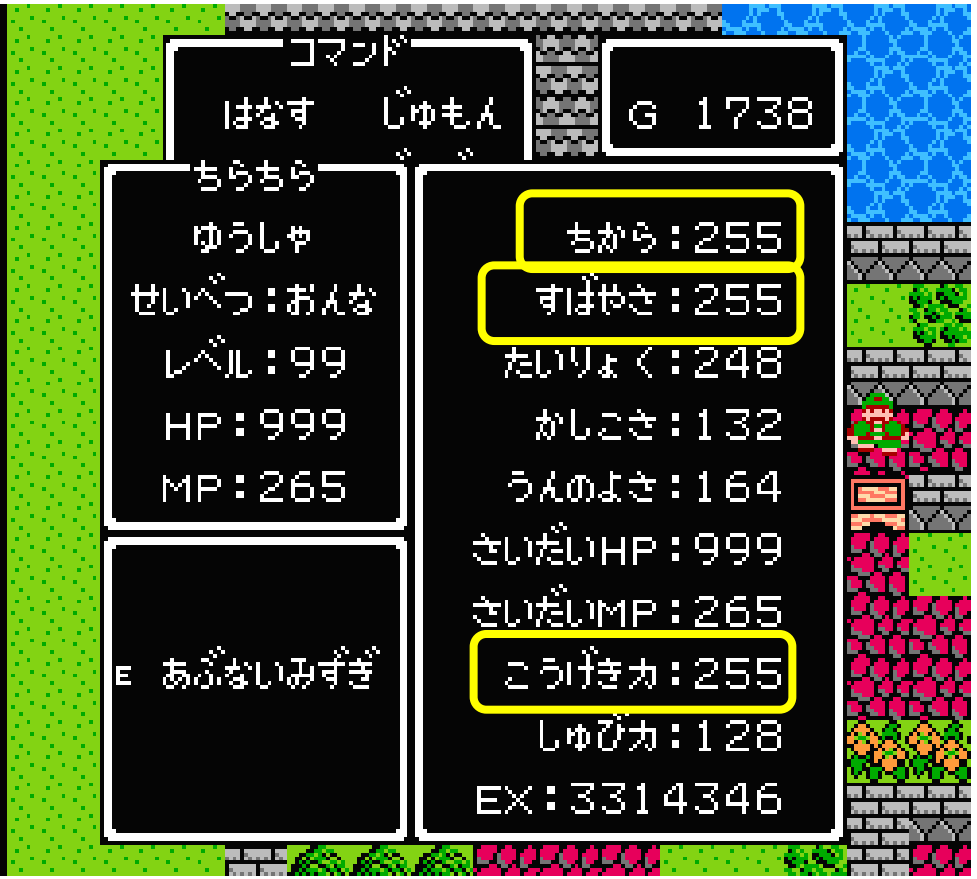
例えば16ビットの場合,

10010001 10111011

MSB: Most Significant Bit (最上位ビット)

LSB: Least Significant Bit (最下位ビット)

ドラゴンクエスト（エニックス, 1986）



ドラゴンクエストIII そして伝説へ…
（エニックス, 1988）

オーバフロー（あふれ）

例えば桁数が 8 ビットに限られている場合:

$$
\begin{array}{rll}
 & 111 & \\
 & 01111010_{(2)} & = 122_{(10)} \\
+) & 10110101_{(2)} & = 181_{(10)} \\
\hline
 & \backslash 100101011_{(2)} & = 43_{(10)}
\end{array}
$$

桁数制限がなければ 303 になるはずだが，$2^8 = 256$ で割った余り 43 が得られる

$$
\begin{array}{rll}
 & 0\ 0 & \\
 & \backslash 01111010_{(2)} & = 122_{(10)} \\
-) & 10110101_{(2)} & = 181_{(10)} \\
\hline
 & \backslash 11000101_{(2)} & = 197_{(10)}
\end{array}
$$

桁数制限がなければ – 59 になるはずだが，$(2^8 - 59)$ を 2^8 で割った余り 197 が得られる

一般に，オーバフローが起きると情報が一部失われる．C 言語では，符号なし数の場合は上記の例のように 2^n を法として合同な数（= あふれた上位ビットを捨てた数）を返すと定められている

シフト演算の場合

```
unsigned short a, b, x, y; // 16 bit

a = 0x1234; // 0001 0010 0011 0100
x = a >> 2; // 0000 0100 1000 1101

b = 0xcafe; // 1100 1010 1111 1110
y = b << 3; // 0101 0111 1111 0000
```

- シフトによって空いたビットには 0 を詰める
- オーバフローが生じた場合，2^k 倍にはならない

余談: 4ギガバイトの壁

32ビットプロセッサ（のほとんど）は，32ビット数を一度に操作することができ，メモリアドレスも 32 ビット数で表す．

64ビットプロセッサ（のほとんど）は，64ビット数を一度に操作することができ，メモリアドレスも 64 ビット数で表す．

メモリアドレスを 32 ビットで表現するプロセッサは，
4,294,967,296バイト ≒ 4.3 ギガバイト までのメモリしか扱うことができない．(64ビット環境への移行が進んだ大きな理由)

例題

ファミリーコンピュータ用ゲーム「ドラゴンクエストIV 導かれし者たち」((株)エニックス, 1990年) には，プレイヤがコインを購入し，スロットマシンやポーカーなどのゲームにそのコインを賭けることのできる「カジノ」と呼ばれるイベントが用意されていた.

コイン1枚は20ゴールド (ゴールドはゲーム世界における通貨単位) で購入できたが，838861枚を指定して購入すると合計わずか4ゴールドで買えてしまうという現象が生じた．このとき，内部でどのような処理が行われていたか推測して述べよ.

(2009年度期末試験)

http://www.youtube.com/watch?v=ajvMdKAnkJ8

解答例

20 [ゴールド/枚] × 838861 [枚] = 16777220 [ゴールド]

きっと有限ビット長表現によるオーバフローが原因であろうと推測してみる.

$2^8 = 256$
$2^{16} = 65536$ (256 × 256)
$2^{24} = 16777216$ (256 × 65536) ← !!
$2^{32} = 4294967296$ (65536 × 65536)

コインの対価を計算する際に 24 ビット長で計算が行われており,
オーバフローを考慮しない処理をしていたため,
16777220 – 16777216 = 4 [ゴールド] で買えたと考えられる.

練習問題

1. 2進数 00101100 を 10進数，16進数で表せ

2. 10進数 123 を 2 進数で表せ

3. 8ビット符号なし2進数 11011011 と 01110010 を加算した結果を8ビット符号なし2進数として表わせ．オーバフローの処理は C 言語で定められている通りとする．

解答例

1. $00101100_{(2)} = (32 + 8 + 4)_{(10)} = 44_{(10)}$
   $= 2c_{(16)}$

2. $123_{(10)} = (64 + 32 + 16 + 8 + 2 + 1)_{(10)} = 01111011_{(2)}$

3. 筆算をするなら，9ビットめへの桁上りを無視すればよい

$$\begin{array}{rr} & 11011011 \\ +) & 01110010 \\ \hline & 101001101 \end{array}$$

(答) 01001101

2進数の筆算が嫌いなら10進数に直して考えてもよい.
$219 + 114 = 333 \equiv 77 \pmod{256} = 64 + 8 + 4 + 1$
$= 01001101_{(2)}$