2進数 (2)

負の数の表現

素朴な方法 (符号と絶対値表示)

通常我々は，10進数の絶対値の前に － をつけて負の数を表す．
同様に，最上位ビットで符号を表し，残りで絶対値を表せばよい

10111011

符号ビット:　　絶対値
0: 正
1: 負

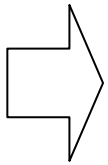
問題点:
- ゼロの表現が2通り存在する
- 加減算が煩雑になる

→ 通常は，2の補数表示と呼ばれる方式が用いられる

「2の補数表示」による「符号つき数」

3ビットの場合:

2進のビット列	符号なし数	2の補数表示による符号つき数
111	7	-1
110	6	-2
101	5	-3
100	4	-4
011	3	3
010	2	2
001	1	1
000	0	0



- 一般にnビットの場合，
  -2^{n-1} ～ $(2^{n-1}-1)$
  の範囲の数を表せる
- MSBが1であれば，負の数である

C言語では，各種の整数型に符号なし，符号つきの種類がある.

signed int, unsigned int
signed short, unsigned short
signed char, unsigned char

000から1ずつ減らして行ったときの表現を素直に考えるとよい

2の補数の定義

n ビットの 2 進数において，ある数 x の
「2の補数 (2's complement)」とは，

$2^n - x$

である
(8 ビットなら，256 – x になる)

2の補数表示による n ビットの符号つき数とは，

- 非負の数 x ($0 \leqq x \leqq 2^{n-1} - 1$) を x の符号なし2進表示で，
- 負の数 –x ($0 < x \leqq 2^{n-1}$) を x の「2の補数」，すなわち ($2^n - x$) の符号なし2進表示で

表したものである

2の補数が使われる理由

符号を気にせず加算・減算を実行することができる

$$
\begin{array}{rll}
 & 01111010 & = 122_{(10)} \\
+) & 11111111 & = -1_{(10)} \\
\hline
 & 101111001 & = 121_{(10)}
\end{array}
$$

はみ出したビットは捨てる

なぜこのようにうまく行くのか？ → 循環しているのがミソ

8ビットの場合，2^8を足すと一巡して元の数に戻る

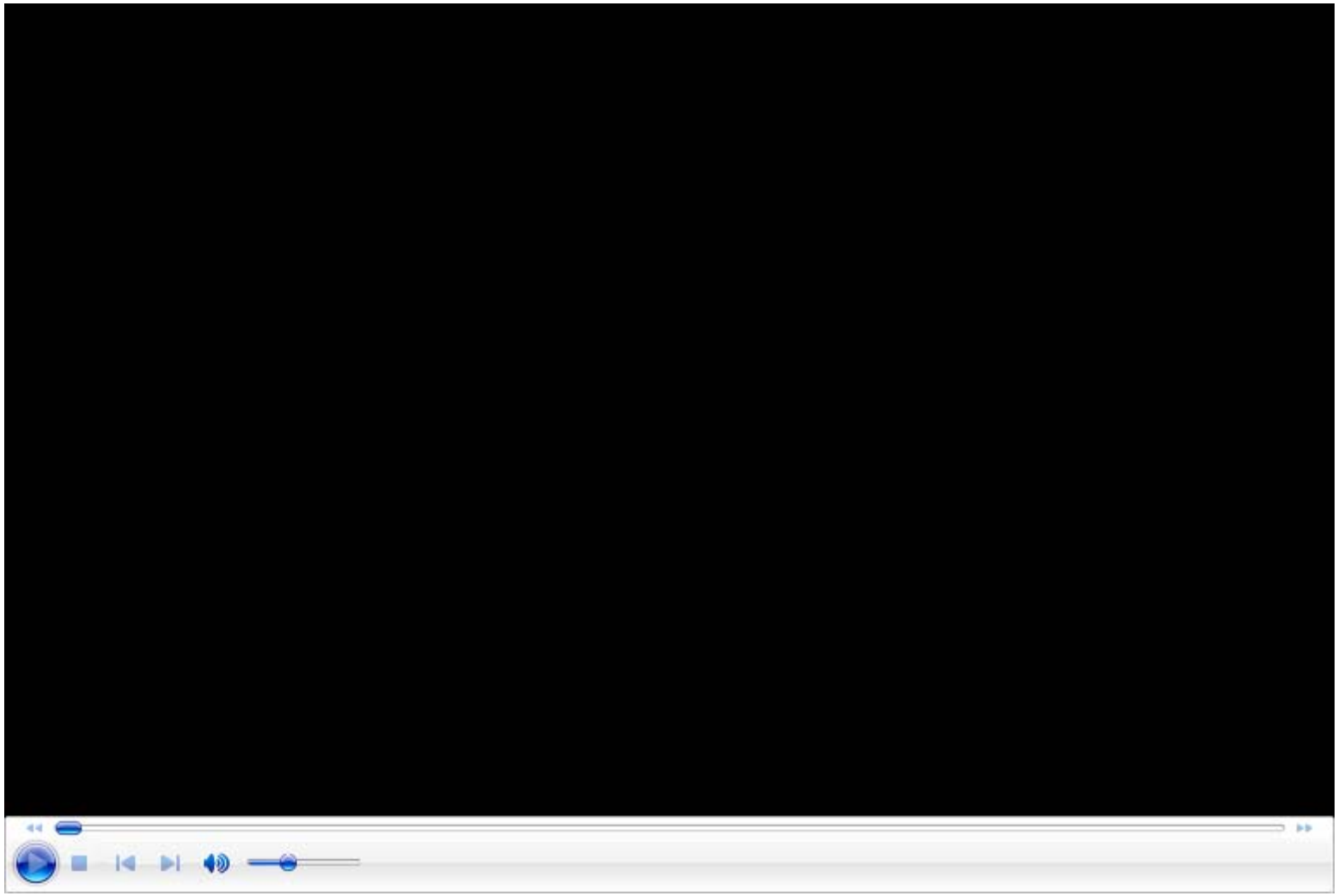
$2^8 - 1$ を足すと，元の数より 1 少ない数に落ち着く

$2^8 - x$ を足すと，元の数より x 少ない数に落ち着く（= 減算）

（符号と絶対値表現だと，数の並ぶ順序が変わってしまうのでこうはいかない:

0, 1, 2, …, 126, 127, -0, -1, -2, …, -126, -127)

「無限1up」



http://www.youtube.com/watch?v=s51UVVMAUkM

2の補数表示の符号つき数の操作

- 加減算
  - 加算はそのまま計算するだけだということが分かった
  - 減算は，『引く数」に負号をつけて2の補数で表し，加算を実行すればよい: 100 – 30 = 100 + (-30)
  - では，符号反転を行うには?

- 符号反転
  - 「ビットを反転して1を足す」

- 2の補数表示 と 普通の10進正負の数の相互変換

符号反転

$x = 30_{(10)} \rightarrow 00011110_{(2)}$

$-x = -30_{(10)} \rightarrow ?_{(2)}$

x の2の補数 $= 2^n - x = (2^n - 1) - x + 1$

111…111（1がn個並んだもの）

```
   11111111   ←  繰り下がりが起きない!
-) 00011110
   --------
   11100001   ←  結局 1 と 0 を反転させるだけ
                 (「1の補数」と呼ばれる)
   11100001
+) 00000001   ←  それに1を足すと，符号反転結
   --------      果が得られる
   11100010
```

結論: 符号反転をするには，各ビットを反転し，1を加えればよい

正→負，負→正 のどちらでもOK （∵$2^n - (2^n - x) = x$）

10進の正負の数との変換

10進 → 2進:

- 絶対値を2進数で表現
  - 正の数であれば，それで終わり
  - 負の数であれば，符号反転処理

2進 → 10進:

- MSBが0か1か?
  - 0であれば非負なので，そのまま10進数へ変換
  - 1であれば負なので，
    - 符号反転処理をしてから10進数へ変換し，負号をつける
    - または，まず10進数へ変換して，それを2^nから引いて，負号をつける

例題

1. 10010110 (2進8ビット，2の補数表示の符号つき数) を10進数に変換せよ

2. –50 (10進数) を 8 ビットの2の補数表示の符号つき2進数で表せ.

解答例

1. MSB = 1なので負の数である.
方法1) まず符号反転処理してから10進に変換し，負号をつける

$$
\begin{array}{ll}
 & 10010110 \\
\rightarrow & 01101001 \\
+) & \qquad\quad 1 \\
\hline
 & 01101010 \rightarrow 106 \rightarrow -106
\end{array}
$$

方法2) まず10進数に変換して，それを256から引いて負号をつける

10010110（正数だと思って変換）
→ 150
→ −(256−150)= −106

2. 50 を符号反転する. $50 = 32 + 16 + 2 = 00110010_{(2)}$ なので,
11001101 + 1 = 11001110

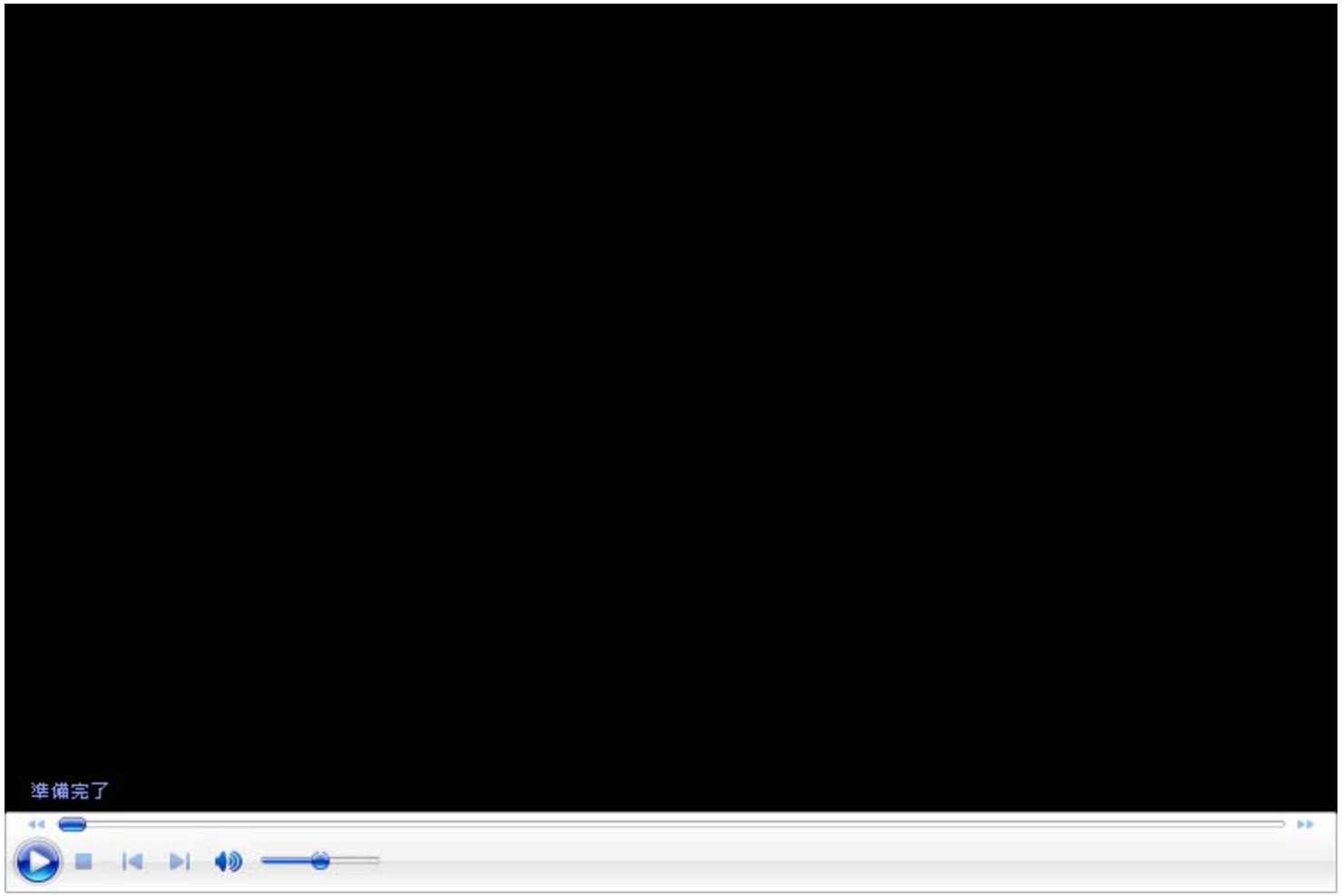
練習問題

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)

「カジノ」



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

練習問題解答例

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.