memo

ディジタル電子回路

授業開始までしばらくお待ちください。

オンライン視聴できない人へ。

オンラインで受講する人も基本的に一緒です。

自宅ネットワークの事情により、授業のストリーミング配信の視聴が困難な学生は以下の対応をしてください。

1. この授業のスライドをよく読んで、不明な点は自分で調べるなどして、わかる範囲で内容を理解する。
2. このページも含め、**必要な部分がすべて理解できたと思うまで以下の 2 ステップを繰り返す**。
   - わからない部分を e-mail 等で質問する。(宛先は `hiroyuki.kobayashi@oit.ac.jp`)
   - e-mail 等による返信をよく読んで理解する。
3. この資料の末尾にある課題を行い、この資料内の方法で (Google Forms で) 提出する。

授業の受講に関して

- 講義資料（スライド等）は**COMMON**に置く。
- 講義は**Google Meet** で行い、録画した講義は**Goole Drive**に置く。

  https://stream.meet.google.com/stream/1d1866da-5bff-4881-96b2-3745413fe31a

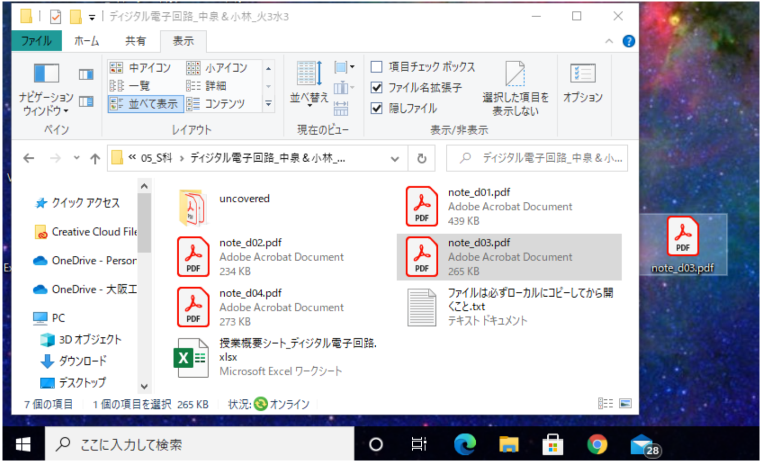
  https://drive.google.com/drive/folders/1bT-z3ICQyMYC_5JvlL29UZYqbOhVG492
- 出席確認レポートは**Google Forms** で提出。(毎回同一 URL)

  https://forms.gle/9ruwtfJg5LQgQNpU7
- **Slack**を補助的な連絡チャネルとする。必須ではないので使いたくなければ使わなくてもいい。授業に関連したちょっとした（重要でない）追加説明をする。気楽な質問手段としても活用されたい。登録は大学の e-mail アドレスで行うこと。

  https://oitkobayashi.slack.com

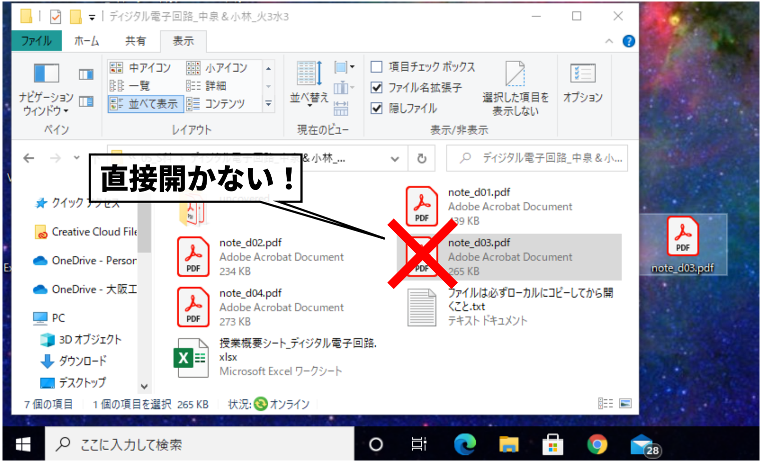
COMMON フォルダの注意事項 (全授業共通)

根源的に悪いのは Windows の仕様なのですが、ご協力ください。



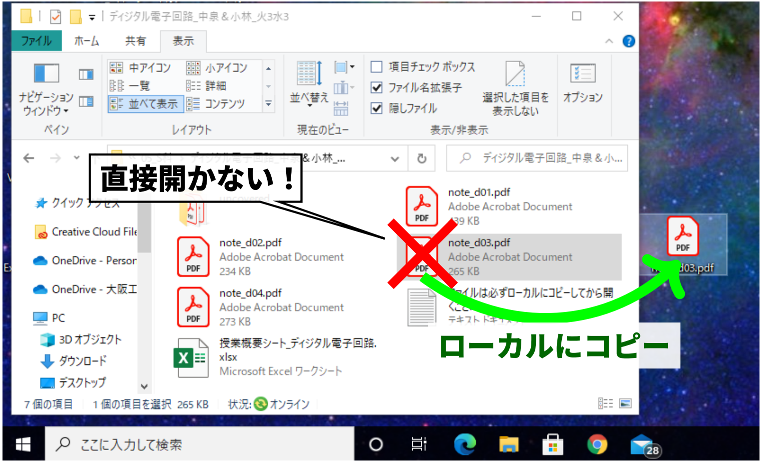
COMMON フォルダの注意事項 (全授業共通)

根源的に悪いのは Windows の仕様なのですが、ご協力ください。



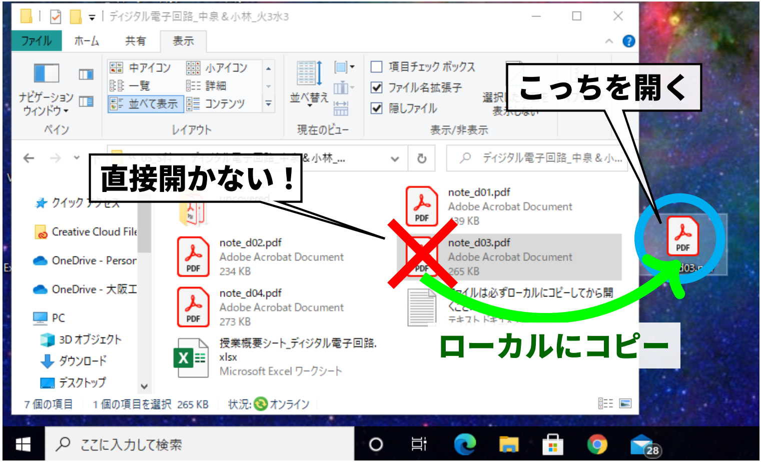
COMMON フォルダの注意事項 (全授業共通)

根源的に悪いのは Windows の仕様なのですが、ご協力ください。



COMMON フォルダの注意事項 (全授業共通)

根源的に悪いのは Windows の仕様なのですが、ご協力ください。



R/S 科ディジタル電子回路
Digital Electronics
『メモリと 2 進数計算』

Google Meet

小林裕之・中泉文孝

大阪工業大学 RD 学部システムデザイン工学科・ロボット工学科

OSAKA INSTITUTE OF TECHNOLOGY

12 of 14

a LaTeX + Beamer slideshow

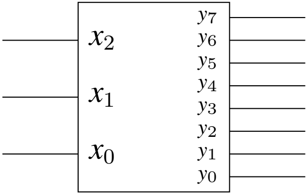
メモリ

デコーダ（decoder）

本来の意味は「復号器[1]」だけど、ここでは特に、

- n **ビットの二進数入力**と 2^n **本の出力**を持ち、
- 「(二進数の値) 番目」の出力だけが 1 になる回路

を考える。



↓二進数↓			7	6	5	4	3	2	1	0
x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

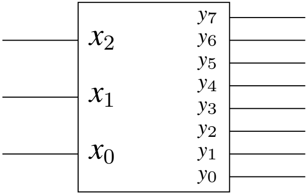
[1] 符号化された信号をもとに戻すもの。

デコーダ（decoder）

本来の意味は「復号器[1]」だけど、ここでは特に、

- n **ビットの二進数入力**と 2^n **本の出力**を持ち、
- 「(二進数の値) 番目」の出力だけが 1 になる回路

を考える。

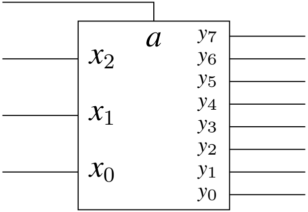


↓二進数↓			7	6	5	4	3	2	1	0
x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0	**1**
0	0	1	0	0	0	0	0	0	**1**	0
0	1	0	0	0	0	0	0	**1**	0	0
0	1	1	0	0	0	0	**1**	0	0	0
1	0	0	0	0	0	**1**	0	0	0	0
1	0	1	0	0	**1**	0	0	0	0	0
1	1	0	0	**1**	0	0	0	0	0	0
1	1	1	**1**	0	0	0	0	0	0	0

[1] 符号化された信号をもとに戻すもの。

デマルチプレクサ（demultiplexer）

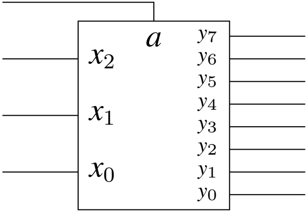
- マルチプレクサの逆
- 入力 (a) を “1” に固定すれば、 ________ そのもの



↓二進数↓			7	6	5	4	3	2	1	0
x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

デマルチプレクサ（demultiplexer）

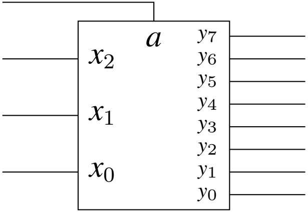
- マルチプレクサの逆
- 入力 (a) を “1” に固定すれば、デコーダ そのもの



↓	二進数	↓	7	6	5	4	3	2	1	0
x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

デマルチプレクサ（demultiplexer）

- マルチプレクサの逆
- 入力 (a) を “1” に固定すれば、デコーダ そのもの



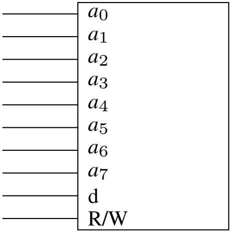
↓二進数↓			7	6	5	4	3	2	1	0
x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	0	0	0	0	0	0	0	0	a
0	0	1	0	0	0	0	0	0	a	0
0	1	0	0	0	0	0	0	a	0	0
0	1	1	0	0	0	0	a	0	0	0
1	0	0	0	0	0	a	0	0	0	0
1	0	1	0	0	a	0	0	0	0	0
1	1	0	0	a	0	0	0	0	0	0
1	1	1	a	0	0	0	0	0	0	0

メモリ (memory)

メモリとは…

- 2^n 個の “0” か “1” を記憶できる箱（セル）を持ち、
- 任意のセルに対して読み込み（や、書き込み）ができる

回路。セルの指定は n ビットの二進数で行うが、これを『　　　　　』と言う。読み込みのみできるものを　　　(　　　　　　　　　　)、読み書きできるものを　　　(　　　　　　　　) という (厳密には違うけど)。



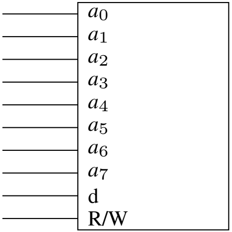
- 左図は 256×1 b のメモリの例
- アドレスは $(00000000)_2$ 番地 (0 番地) から $(11111111)_2$ 番地 (255 番地) まで
- $a_0 \sim a_7$ の信号で　　　　　を指定すると、該当するセルが d 信号としてアクセスできる
- R/W 信号で「読み出し」と「書き込み」を切り替える RAM (のつもりの絵)

メモリ (memory)

メモリとは…

- 2^n 個の “0” か “1” を記憶できる箱（セル）を持ち、
- 任意のセルに対して読み込み（や、書き込み）ができる

回路。セルの指定は n ビットの二進数で行うが、これを『**アドレス**』と言う。読み込みのみできるものを　　(　　　　　　　　)、読み書きできるものを　　(　　　　　　　　) という (厳密には違うけど)。



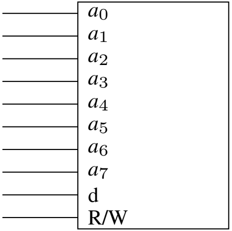
- 左図は 256 × 1 b のメモリの例
- アドレスは $(00000000)_2$ 番地 (0 番地) から $(11111111)_2$ 番地 (255 番地) まで
- $a_0 \sim a_7$ の信号で　　　　　を指定すると、該当するセルが d 信号としてアクセスできる
- R/W 信号で「読み出し」と「書き込み」を切り替える RAM (のつもりの絵)

メモリ (memory)

メモリとは…

- 2^n 個の "0" か "1" を記憶できる箱（セル）を持ち、
- 任意のセルに対して読み込み（や、書き込み）ができる

回路。セルの指定は n ビットの二進数で行うが、これを『**アドレス**』と言う。読み込みのみできるものを ROM (   )、読み書きできるものを   (   ) という (厳密には違うけど)。



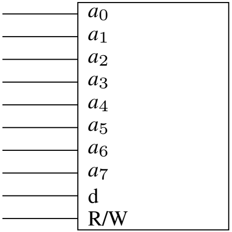
- 左図は 256×1 b のメモリの例
- アドレスは $(00000000)_2$ 番地 (0 番地) から $(11111111)_2$ 番地 (255 番地) まで
- $a_0 \sim a_7$ の信号で   を指定すると、該当するセルが d 信号としてアクセスできる
- R/W 信号で「読み出し」と「書き込み」を切り替える RAM (のつもりの絵)

メモリ (memory)

メモリとは…

- 2^n 個の "0" か "1" を記憶できる箱（セル）を持ち、
- 任意のセルに対して読み込み（や、書き込み）ができる

回路。セルの指定は n ビットの二進数で行うが、これを『**アドレス**』と言う。読み込みのみできるものを ROM (Read Only Memory)、読み書きできるものを　　　(　　　) という (厳密には違うけど)。



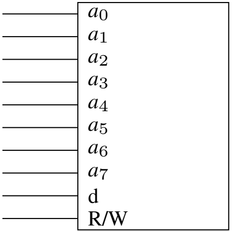
- 左図は 256 × 1 b のメモリの例
- アドレスは $(00000000)_2$ 番地 (0 番地) から $(11111111)_2$ 番地 (255 番地) まで
- $a_0 \sim a_7$ の信号で　　　を指定すると、該当するセルが d 信号としてアクセスできる
- R/W 信号で「読み出し」と「書き込み」を切り替える RAM (のつもりの絵)

メモリ (memory)

メモリとは…

- 2^n 個の “0” か “1” を記憶できる箱（セル）を持ち、
- 任意のセルに対して読み込み（や、書き込み）ができる

回路。セルの指定は n ビットの二進数で行うが、これを『**アドレス**』と言う。読み込みのみできるものを ROM (Read Only Memory)、読み書きできるものを RAM (
) という (厳密には違うけど)。



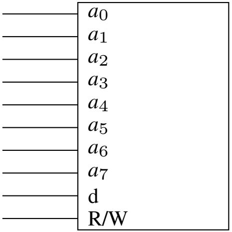
- 左図は 256 × 1 b のメモリの例
- アドレスは $(00000000)_2$ 番地 (0 番地) から $(11111111)_2$ 番地 (255 番地) まで
- $a_0 \sim a_7$ の信号で　　　　　を指定すると、該当するセルが d 信号としてアクセスできる
- R/W 信号で「読み出し」と「書き込み」を切り替える RAM (のつもりの絵)

メモリ (memory)

メモリとは…

- 2^n 個の "0" か "1" を記憶できる箱（セル）を持ち、
- 任意のセルに対して読み込み（や、書き込み）ができる

回路。セルの指定は n ビットの二進数で行うが、これを『**アドレス**』と言う。読み込みのみできるものを ROM (Read Only Memory)、読み書きできるものを RAM (Random Access Memory) という (厳密には違うけど)。



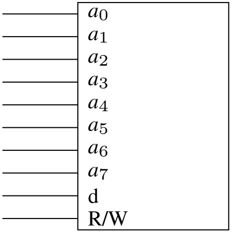
- 左図は 256 × 1 b のメモリの例
- アドレスは $(00000000)_2$ 番地 (0 番地) から $(11111111)_2$ 番地 (255 番地) まで
- $a_0 \sim a_7$ の信号で　　　を指定すると、該当するセルが d 信号としてアクセスできる
- R/W 信号で「読み出し」と「書き込み」を切り替える RAM (のつもりの絵)

メモリ (memory)

メモリとは…

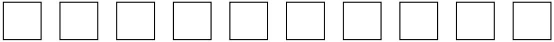
- 2^n 個の “0” か “1” を記憶できる箱（セル）を持ち、
- 任意のセルに対して読み込み（や、書き込み）ができる

回路。セルの指定は n ビットの二進数で行うが、これを『**アドレス**』と言う。読み込みのみできるものを ROM (Read Only Memory)、読み書きできるものを RAM (Random Access Memory) という (厳密には違うけど)。



- 左図は 256 × 1 b のメモリの例
- アドレスは $(00000000)_2$ 番地 (0 番地) から $(11111111)_2$ 番地 (255 番地) まで
- $a_0 \sim a_7$ の信号でアドレスを指定すると、該当するセルが d 信号としてアクセスできる
- R/W 信号で「読み出し」と「書き込み」を切り替える RAM (のつもりの絵)

メモリの物理的な配置（ROM・RAM 共通!）

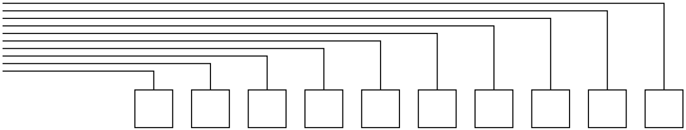


こう配置すると n 個のセルに対して $\mathcal{O}(\quad)$本の選択線が必要。

こうすれば $\mathcal{O}(\qquad)$本の配線で済む!

というわけで、現実のメモリはこういう構造になっています。

メモリの物理的な配置（ROM・RAM 共通!）

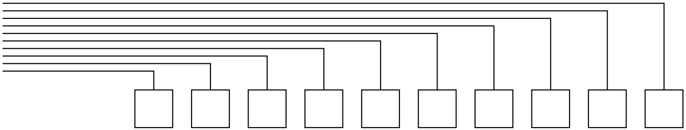


こう配置すると n 個のセルに対して $\mathcal{O}(\quad)$本の選択線が必要。

こうすれば $\mathcal{O}(\qquad)$本の配線で済む!

というわけで、現実のメモリはこういう構造になっています。

メモリの物理的な配置（ROM・RAM 共通!）

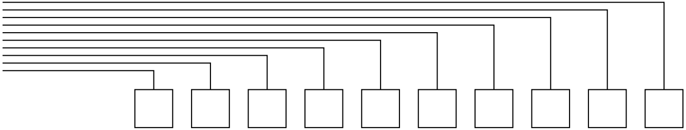


こう配置すると n 個のセルに対して $\mathcal{O}(n)$ 本の選択線が必要。

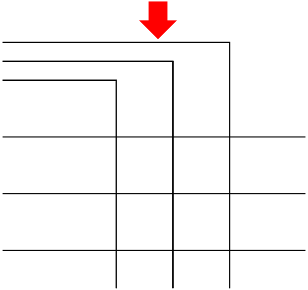
こうすれば $\mathcal{O}(\quad)$ 本の配線で済む!

というわけで、現実のメモリはこういう構造になっています。

メモリの物理的な配置（ROM・RAM 共通!）



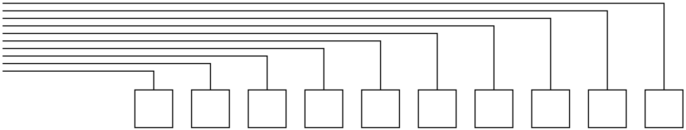
こう配置すると n 個のセルに対して $\mathcal{O}(n)$ 本の選択線が必要。



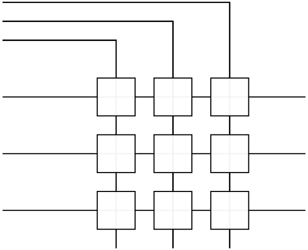
こうすれば $\mathcal{O}(\quad)$ 本の配線で済む!

というわけで、現実のメモリはこういう構造になっています。

メモリの物理的な配置（ROM・RAM 共通!）



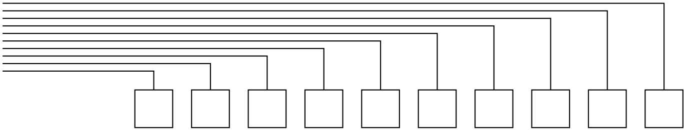
こう配置すると n 個のセルに対して $\mathcal{O}(n)$本の選択線が必要。



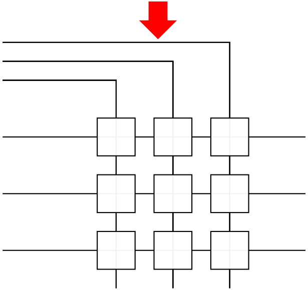
こうすれば $\mathcal{O}(\quad)$本の配線で済む!

というわけで、現実のメモリはこういう構造になっています。

メモリの物理的な配置（ROM・RAM 共通!）



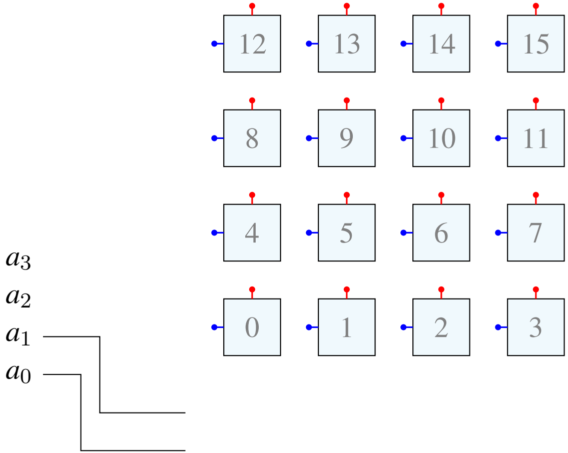
こう配置すると n 個のセルに対して $\mathcal{O}(n)$ 本の選択線が必要。



こうすれば $\mathcal{O}(\sqrt{n})$ 本の配線で済む!

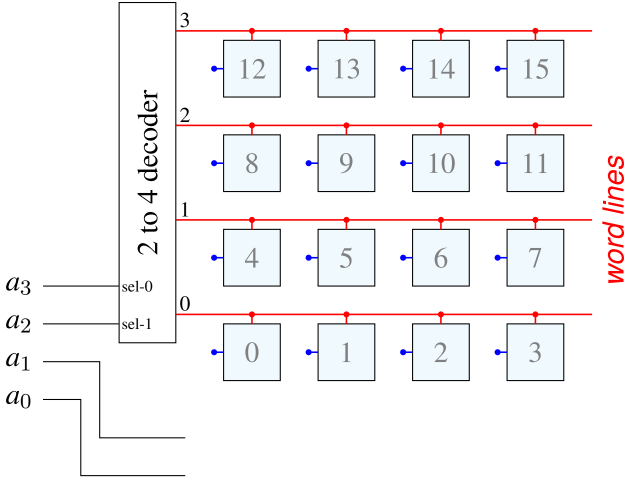
というわけで、現実のメモリはこういう構造になっています。

メモリの構造（ROM・RAM 共通!）



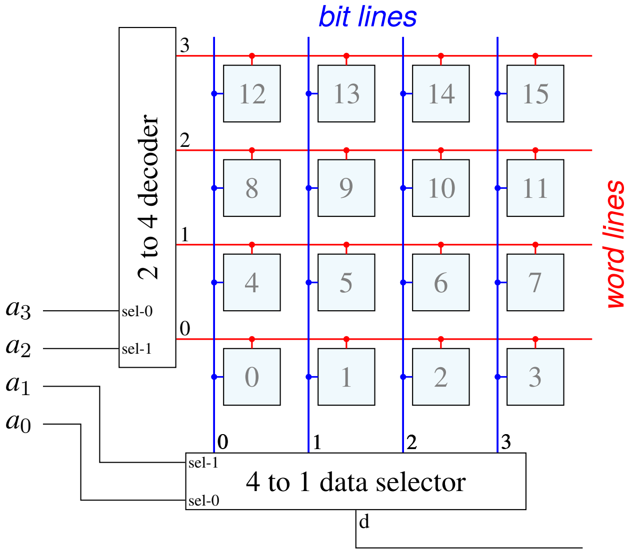
- アドレスを半分に分け、
- MSB 側を　　　　　の選択に用い、LSB 側を　　　　　の選択に用いる（逆でも良い）。
- 各セルは　　　　　で起動して、　　　　　で 0/1 情報を出し入れする。

メモリの構造（ROM・RAM 共通!）



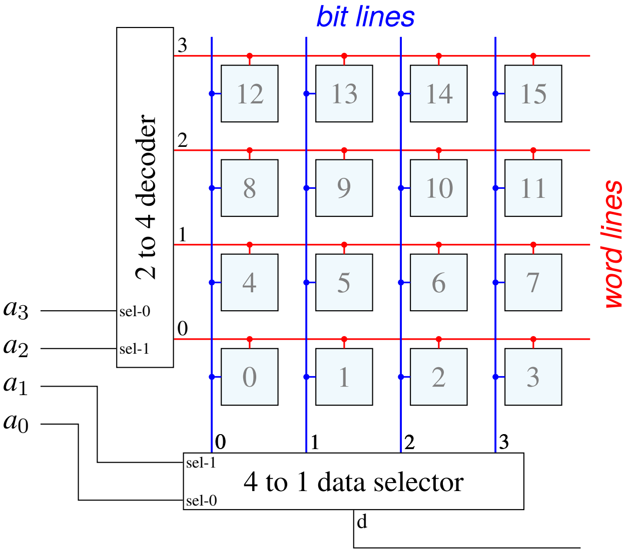
- アドレスを半分に分け、
- MSB 側をワード線の選択に用い、LSB 側を　　　　　の選択に用いる（逆でも良い）。
- 各セルは　　　　　で起動して、　　　　　で 0/1 情報を出し入れする。

メモリの構造（ROM・RAM 共通!）



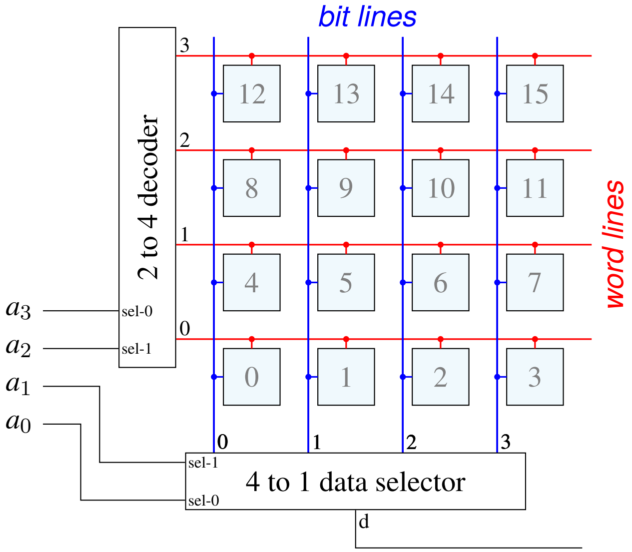
- アドレスを半分に分け、
- MSB 側をワード線の選択に用い、LSB 側をビット線の選択に用いる（逆でも良い）。
- 各セルは　　　　　で起動して、　　　　　で 0/1 情報を出し入れする。

メモリの構造（ROM・RAM 共通!）



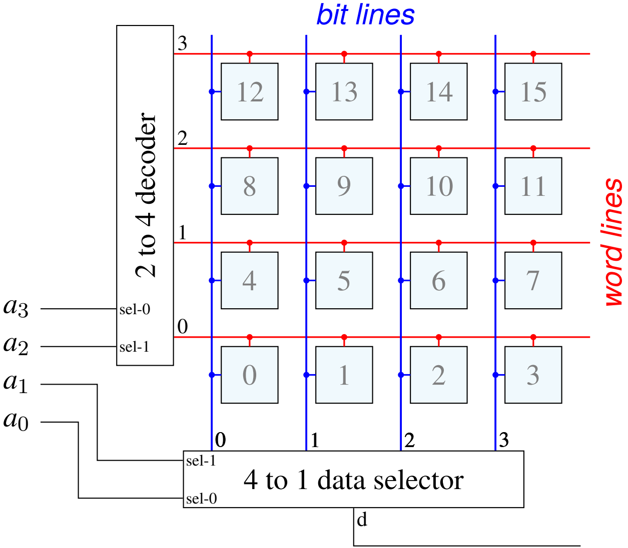
- アドレスを半分に分け、
- MSB 側をワード線の選択に用い、LSB 側をビット線の選択に用いる（逆でも良い）。
- 各セルはワード線で起動して、で 0/1 情報を出し入れする。

メモリの構造（ROM・RAM 共通!）

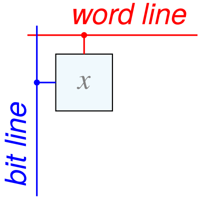


- アドレスを半分に分け、
- MSB 側をワード線の選択に用い、LSB 側をビット線の選択に用いる（逆でも良い）。
- 各セルはワード線で起動して、ビット線で 0/1 情報を出し入れする。

メモリの構造（ROM・RAM 共通!）

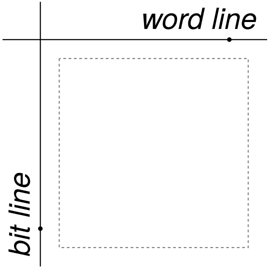


- アドレスを半分に分け、
- MSB 側をワード線の選択に用い、LSB 側をビット線の選択に用いる（逆でも良い）。
- 各セルはワード線で起動して、ビット線で 0/1 情報を出し入れする。

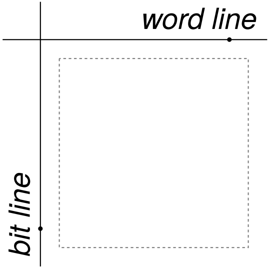


← どう作る？

メモリセルの基本的な発想



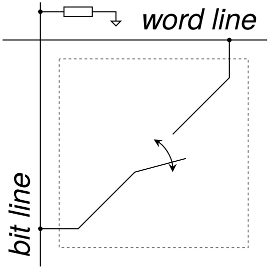
かなり思い切った概念図



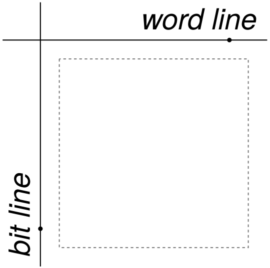
いくらか実際に近い図

- ワード線とビット線の間にスイッチを置き、
- 閉じれば“ ”を記録、開けば“ ”を記録。
- 製造時に on/off を決めて作る →

メモリセルの基本的な発想



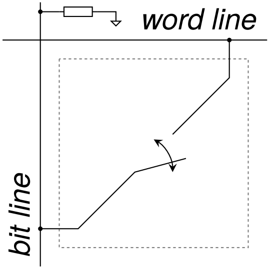
かなり思い切った概念図



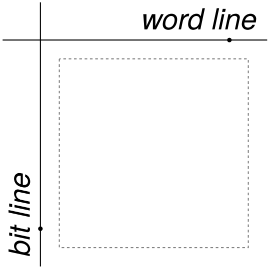
いくらか実際に近い図

- ワード線とビット線の間にスイッチを置き、
- 閉じれば“ ”を記録、開けば“ ”を記録。
- 製造時に on/off を決めて作る →

メモリセルの基本的な発想



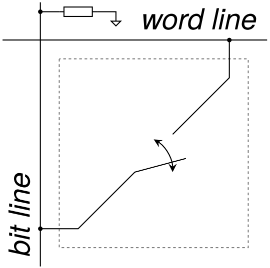
かなり思い切った概念図



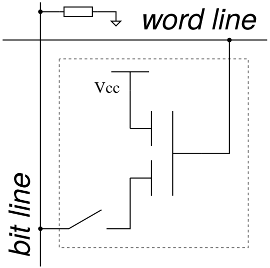
いくらか実際に近い図

- ワード線とビット線の間にスイッチを置き、
- 閉じれば “1” を記録、開けば “0” を記録。
- 製造時に on/off を決めて作る →

メモリセルの基本的な発想



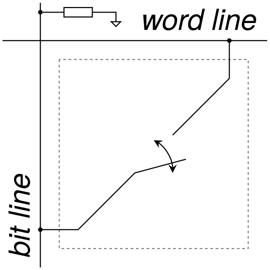
かなり思い切った概念図



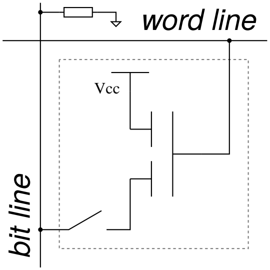
いくらか実際に近い図

- ワード線とビット線の間にスイッチを置き、
- 閉じれば “1” を記録、開けば “0” を記録。
- 製造時に on/off を決めて作る →

メモリセルの基本的な発想



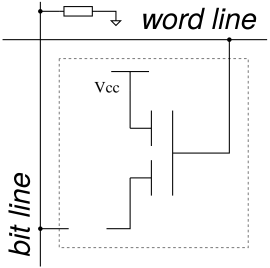
かなり思い切った概念図



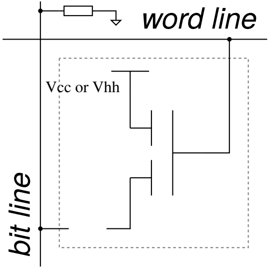
いくらか実際に近い図

- ワード線とビット線の間にスイッチを置き、
- 閉じれば "1" を記録、開けば "0" を記録。
- 製造時に on/off を決めて作る →**マスク ROM**

一回だけ『焼ける』



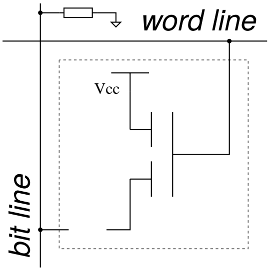
製造時に値が決まるマスク ROM (0 を作り込んだ例)



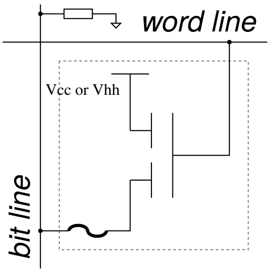
一度だけ書き込める

- FET のビット線側に　　　　　を作りこんでおき (値 =1)、
- Vcc よりも高い Vhh を印加した状態で、
- ワード線 =1, ビット線 =0 とすれば過電流でその　　　が焼き切れ、“0” が書き込まれる。

一回だけ『焼ける』



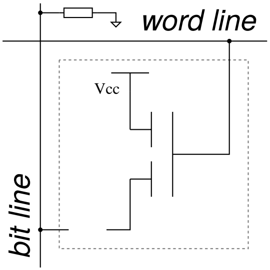
製造時に値が決まるマスク ROM (0 を作り込んだ例)



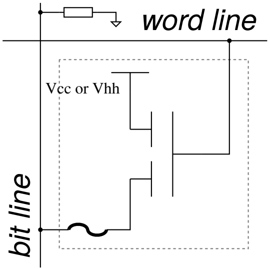
一度だけ書き込める

- FET のビット線側に　　　　　　を作りこんでおき (値 =1)、
- Vcc よりも高い Vhh を印加した状態で、
- ワード線 =1, ビット線 =0 とすれば過電流でその　　　が焼き切れ、“0” が書き込まれる。

一回だけ『焼ける』



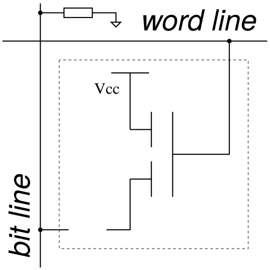
製造時に値が決まるマスク ROM (0 を作り込んだ例)



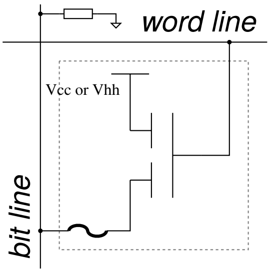
一度だけ書き込める

- FET のビット線側にフューズを作りこんでおき (値 =1)、
- Vcc よりも高い Vhh を印加した状態で、
- ワード線 =1, ビット線 =0 とすれば過電流でその fuseが焼き切れ、"0" が書き込まれる。

一回だけ『焼ける』PROM (Programmable ROM)



製造時に値が決まるマスク ROM (0 を作り込んだ例)



一度だけ書き込めるPROM

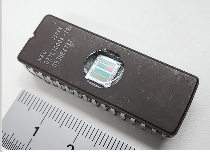
- FET のビット線側にフューズを作りこんでおき (値 =1)、
- Vcc よりも高い Vhh を印加した状態で、
- ワード線 =1, ビット線 =0 とすれば過電流でその fuseが焼き切れ、"0" が書き込まれる。

いろいろな ROM

基本は同じ。どのような仕組みでスイッチを ON/OFF するか。

代表的な ROM

- 【　　　　　　　】…製造時に配線済み。書き込み不可。
- 【　　　　】…電流でヒューズを焼き切る。一度書き込んだら終わり。
- 【　　　　　】…　　　　で全消去できる。
- 【　　　　　　】…**電気的**に 消去できる。フラッシュメモリも広義にはこの一種。



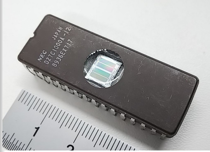
UV-EPROM の例

いろいろな ROM

基本は同じ。どのような仕組みでスイッチを ON/OFF するか。

代表的な ROM

- 【マスク ROM】…製造時に配線済み。書き込み不可。
- 【　　　　】…電流でヒューズを焼き切る。一度書き込んだら終わり。
- 【　　　　　】…　　　　で全消去できる。
- 【　　　　　　　】…**電気的**に 消去できる。フラッシュメモリも広義にはこの一種。



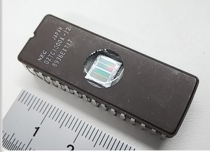
UV-EPROM の例

いろいろな ROM

基本は同じ。どのような仕組みでスイッチを ON/OFF するか。

代表的な ROM

- 【マスク ROM】…製造時に配線済み。書き込み不可。
- 【PROM】…電流でヒューズを焼き切る。一度書き込んだら終わり。
- 【　　　】…　　　で全消去できる。
- 【　　　　】…**電気的**に 消去できる。フラッシュメモリも広義にはこの一種。



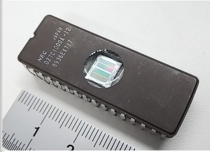
UV-EPROM の例

いろいろな ROM

基本は同じ。どのような仕組みでスイッチを ON/OFF するか。

代表的な ROM

- **【マスク ROM】**…製造時に配線済み。書き込み不可。
- **【PROM】**…電流でヒューズを焼き切る。一度書き込んだら終わり。
- **【EPROM】**…　　　で全消去できる。
- **【　　　　】**…**電気的**に 消去できる。フラッシュメモリも広義にはこの一種。

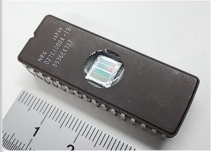


UV-EPROM の例

基本は同じ。どのような仕組みでスイッチを ON/OFF するか。

代表的な ROM

- 【マスク ROM】…製造時に配線済み。書き込み不可。
- 【PROM】…電流でヒューズを焼き切る。一度書き込んだら終わり。
- 【EPROM】…紫外線で全消去できる。
- 【　　　　　　】…**電気的**に 消去できる。フラッシュメモリも広義にはこの一種。



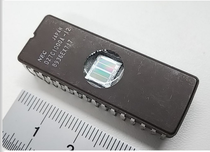
UV-EPROM の例

いろいろな ROM

基本は同じ。どのような仕組みでスイッチを ON/OFF するか。

代表的な ROM

- **【マスク ROM】**…製造時に配線済み。書き込み不可。
- **【PROM】**…電流でヒューズを焼き切る。一度書き込んだら終わり。
- **【EPROM】**…紫外線で全消去できる。
- **【EEPROM】**…**電気的**に 消去できる。フラッシュメモリも広義にはこの一種。



UV-EPROM の例

DRAM 前夜

書き込み可能なメモリには苦労していた。

磁気を使う

- 磁気 (例: Busicom 161) (∼60 年代)
- バブルメモリ (例: FM-8, コナミバブルシステム) (∼80 年代)

SRAM (RAM)

- 基本的には を使って記憶する。
- トランジスタが最低でも 4 つ要るので価格と集積度で不利。
- 速いので今でもたくさん使われている。

DRAM 前夜

書き込み可能なメモリには苦労していた。

磁気を使う

- 磁気コアメモリ (例: Busicom 161) (〜60 年代)
- バブルメモリ (例: FM-8, コナミバブルシステム) (〜80 年代)

SRAM (　　　RAM)

- 基本的には　　　を使って記憶する。
- トランジスタが最低でも 4 つ要るので価格と集積度で不利。
- 速いので今でもたくさん使われている。

DRAM 前夜

書き込み可能なメモリには苦労していた。

磁気を使う

- 磁気コアメモリ (例: Busicom 161) (〜60 年代)
- バブルメモリ (例: FM-8, コナミバブルシステム) (〜80 年代)

SRAM (Static RAM)

- 基本的には　　を使って記憶する。
- トランジスタが最低でも 4 つ要るので価格と集積度で不利。
- 速いので今でもたくさん使われている。

DRAM 前夜

書き込み可能なメモリには苦労していた。

磁気を使う

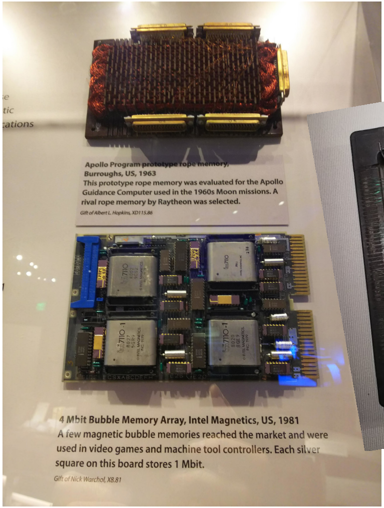
- 磁気コアメモリ (例: Busicom 161) (〜60 年代)
- バブルメモリ (例: FM-8, コナミバブルシステム) (〜80 年代)

SRAM (Static RAM)

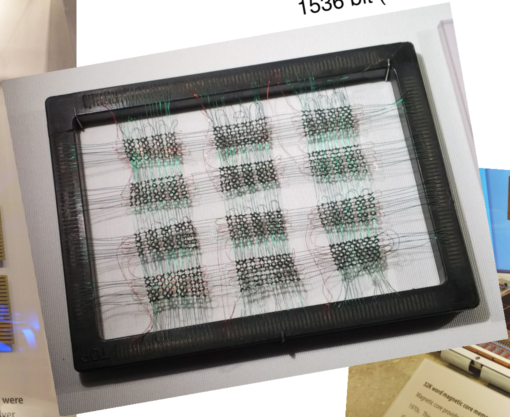
- 基本的には FF を使って記憶する。
- トランジスタが最低でも 4 つ要るので価格と集積度で不利。
- 速いので今でもたくさん使われている。

(参考) コアメモリ〜バブルメモリ (CHM にて撮影)

コアダンプって聞いたことあります？

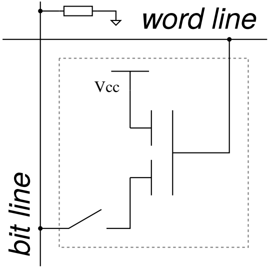


1536 bit (1965)

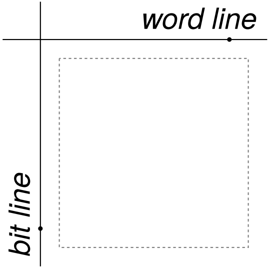


32K word (1978)

真打ち　　　　登場！



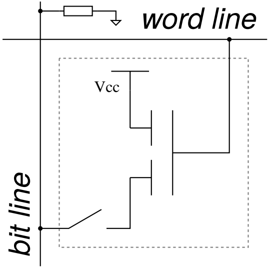
マスク ROM



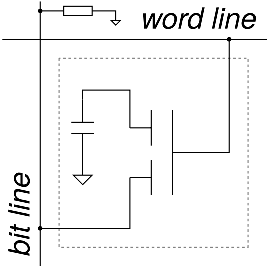
読み書き自在な

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み:　　　を　にして、　　　を　にする。
- “0” の書き込み:　　　を　にして、　　　を　にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



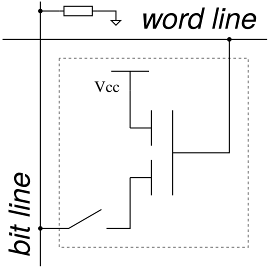
マスク ROM



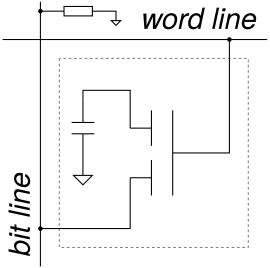
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: を にして、 を にする。
- “0” の書き込み: を にして、 を にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



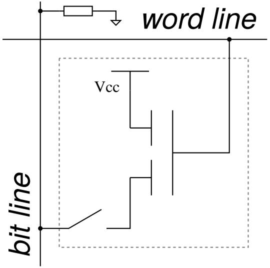
マスク ROM



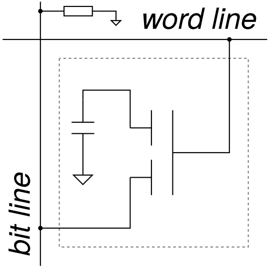
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: を にして、 を にする。
- “0” の書き込み: を にして、 を にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



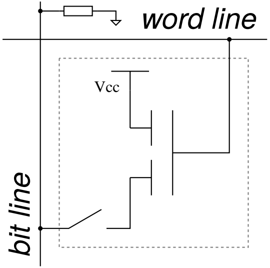
マスク ROM



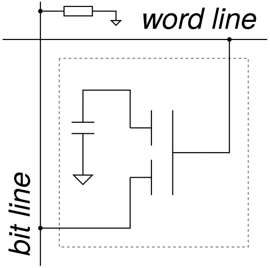
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: ビット線を　にして、　　　　　　を　にする。
- “0” の書き込み:　　　　　　　を　にして、　　　　　　を　にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



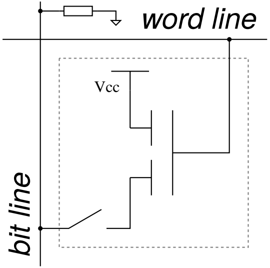
マスク ROM



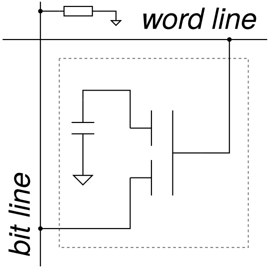
読み書き自在なDRAM

- キャパシタが充電されてれば "1"、されていなければ "0"。
- "1" の書き込み: ビット線を 1 にして、　　　　　を　にする。
- "0" の書き込み: 　　　　　を　にして、　　　　　を　にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



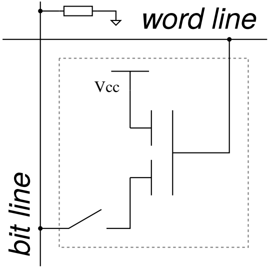
マスク ROM



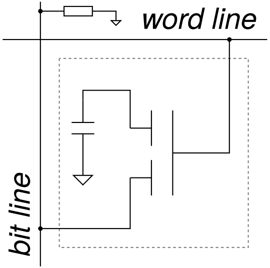
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: ビット線を 1 にして、ワード線を　にする。
- “0” の書き込み: 　　　　を　にして、　　　　を　にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



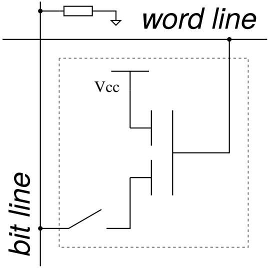
マスク ROM



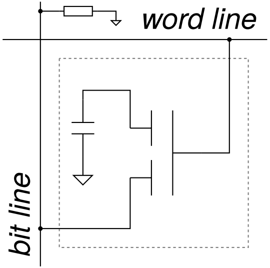
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: ビット線を 1 にして、ワード線を 1 にする。
- “0” の書き込み: を にして、 を にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



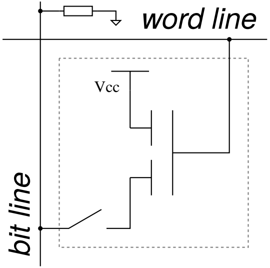
マスク ROM



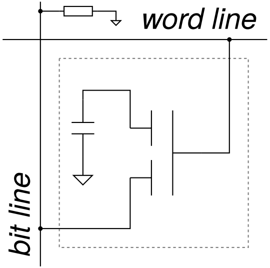
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: ビット線を 1 にして、ワード線を 1 にする。
- “0” の書き込み: ビット線を　にして、　　　　　　を　にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



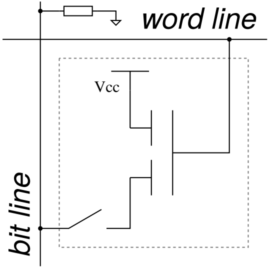
マスク ROM



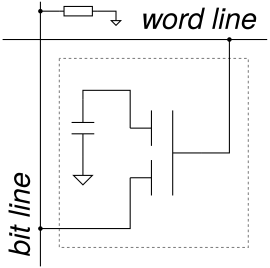
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: ビット線を 1 にして、ワード線を 1 にする。
- “0” の書き込み: ビット線を 0 にして、　　　　を　にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



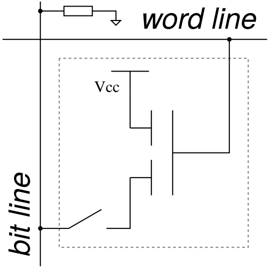
マスク ROM



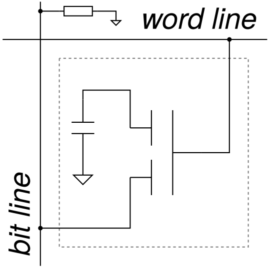
読み書き自在なDRAM

- キャパシタが充電されてれば “1”、されていなければ “0”。
- “1” の書き込み: ビット線を 1 にして、ワード線を 1 にする。
- “0” の書き込み: ビット線を 0 にして、ワード線を　にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

真打ち DRAM (Dynamic RAM) 登場！



マスク ROM



読み書き自在なDRAM

- キャパシタが充電されてれば "1"、されていなければ "0"。
- "1" の書き込み: ビット線を 1 にして、ワード線を 1 にする。
- "0" の書き込み: ビット線を 0 にして、ワード線を 1 にする。
- 注: 実際の DRAM はビット線の向こう側にセンスアンプがあって信号を増幅してから読み出す。

DRAM と SRAM （常識として知っておこう）

- 初の DRAM 製品は　　　が作った。(1Kib の『1103』(1970))
- SRAM は FET を最低 4 つは使うので、DRAM に比べて集積度が低い。
- DRAM は **FET とキャパシタだけ** で作れる。→
- SRAM は DRAM より　　　。
- DRAM は放っておくとキャパシタが放電してデータが失われるため、常にデータの再書き込み（　　　　　　　）をする必要がある[2]。
- SRAM は ↑ その必要がないので、周辺回路が単純で済みついでに低消費電力 (→ 小規模な組み込みにも向く)。

[2] この辺が DRAM の D たる所以なり。

DRAM と SRAM （常識として知っておこう）

- 初の DRAM 製品はIntel が作った。(1Kib の『1103』(1970))
- SRAM は FET を最低 4 つは使うので、DRAM に比べて集積度が低い。
- DRAM は **FET とキャパシタだけ** で作れる。→
- SRAM は DRAM より 。
- DRAM は放っておくとキャパシタが放電してデータが失われるため、常にデータの再書き込み（ ）をする必要がある[2]。
- SRAM は↑その必要がないので、周辺回路が単純で済みついでに低消費電力 (→ 小規模な組み込みにも向く)。

[2] この辺が DRAM の D たる所以なり。

DRAM と SRAM （常識として知っておこう）

- 初の DRAM 製品はIntel が作った。(1Kib の『1103』(1970))
- SRAM は FET を最低 4 つは使うので、DRAM に比べて集積度が低い。
- DRAM は **FET とキャパシタだけ** で作れる。→ 集積度が上げられる・安い
- SRAM は DRAM より　　。
- DRAM は放っておくとキャパシタが放電してデータが失われるため、常にデータの再書き込み（　　　　　　　）をする必要がある[2]。
- SRAM は ↑ その必要がないので、周辺回路が単純で済みついでに低消費電力 (→ 小規模な組み込みにも向く)。

[2] この辺が DRAM の D たる所以なり。

DRAM と SRAM （常識として知っておこう）

- 初の DRAM 製品はIntel が作った。(1Kib の『1103』(1970))
- SRAM は FET を最低 4 つは使うので、DRAM に比べて集積度が低い。
- DRAM は **FET とキャパシタだけ** で作れる。→ 集積度が上げられる・安い
- SRAM は DRAM より速い。
- DRAM は放っておくとキャパシタが放電してデータが失われるため、常にデータの再書き込み（　　　　　　　　）をする必要がある[2]。
- SRAM は ↑ その必要がないので、周辺回路が単純で済みついでに低消費電力 (→ 小規模な組み込みにも向く)。

[2] この辺が DRAM の D たる所以なり。

DRAM と SRAM （常識として知っておこう）

- 初の DRAM 製品はIntel が作った。(1Kib の『1103』(1970))
- SRAM は FET を最低 4 つは使うので、DRAM に比べて集積度が低い。
- DRAM は **FET とキャパシタだけ** で作れる。→ 集積度が上げられる・安い
- SRAM は DRAM より速い。
- DRAM は放っておくとキャパシタが放電してデータが失われるため、常にデータの再書き込み（**リフレッシュ**）をする必要がある[2]。
- SRAM は ↑ その必要がないので、周辺回路が単純で済みついでに低消費電力 (→ 小規模な組み込みにも向く)。

[2]この辺が DRAM の D たる所以なり。

メモリクイズ

問: 次の用途にもっともよく使われているメモリはどれか?

1. PC の主記憶 →
2. スマートフォンの主記憶 →
3. SD カードや USB メモリ →
4. CPU の cache →
5. マイコン (MCU) の内蔵 RAM →

※ それぞれ理由もセットで考えよう。

問: 以下に答えよ。

1. EEPROM は読み書きできるのになぜ **RAM** ではなく、EEP**ROM** なのか?
2. DRAM を初めて発売し、ひと儲けし…いやかなり儲けた企業はどこか?

メモリクイズ

問: 次の用途にもっともよく使われているメモリはどれか?

1. PC の主記憶 → DRAM
2. スマートフォンの主記憶 →
3. SD カードや USB メモリ →
4. CPU の cache →
5. マイコン (MCU) の内蔵 RAM →

※ それぞれ理由もセットで考えよう。

問: 以下に答えよ。

1. EEPROM は読み書きできるのになぜ **RAM** ではなく、EEP**ROM** なのか?
2. DRAM を初めて発売し、ひと儲けし…いやかなり儲けた企業はどこか?

メモリクイズ

問: 次の用途にもっともよく使われているメモリはどれか?

1. PC の主記憶 → DRAM
2. スマートフォンの主記憶 → DRAM
3. SD カードや USB メモリ →
4. CPU の cache →
5. マイコン (MCU) の内蔵 RAM →

※ **それぞれ理由もセットで考えよう。**

問: 以下に答えよ。

1. EEPROM は読み書きできるのになぜ **RAM** ではなく、EEPROM なのか?
2. DRAM を初めて発売し、ひと儲けし…いやかなり儲けた企業はどこか?

メモリクイズ

問: 次の用途にもっともよく使われているメモリはどれか?

1. PC の主記憶 → DRAM
2. スマートフォンの主記憶 → DRAM
3. SD カードや USB メモリ → EEPROM(フラッシュメモリ)
4. CPU の cache →
5. マイコン (MCU) の内蔵 RAM →

※ それぞれ理由もセットで考えよう。

問: 以下に答えよ。

1. EEPROM は読み書きできるのになぜ **RAM** ではなく、EEPROM なのか?
2. DRAM を初めて発売し、ひと儲けし…いやかなり儲けた企業はどこか?

メモリクイズ

問: 次の用途にもっともよく使われているメモリはどれか?

1. PC の主記憶 → DRAM
2. スマートフォンの主記憶 → DRAM
3. SD カードや USB メモリ → EEPROM(フラッシュメモリ)
4. CPU の cache → SRAM
5. マイコン (MCU) の内蔵 RAM →

※ それぞれ理由もセットで考えよう。

問: 以下に答えよ。

1. EEPROM は読み書きできるのになぜ **RAM** ではなく、EEP**ROM** なのか?
2. DRAM を初めて発売し、ひと儲けし…いやかなり儲けた企業はどこか?

メモリクイズ

問: 次の用途にもっともよく使われているメモリはどれか?

1. PC の主記憶 → DRAM
2. スマートフォンの主記憶 → DRAM
3. SD カードや USB メモリ → EEPROM(フラッシュメモリ)
4. CPU の cache → SRAM
5. マイコン (MCU) の内蔵 RAM → SRAM

※ それぞれ理由もセットで考えよう。

問: 以下に答えよ。

1. EEPROM は読み書きできるのになぜ **RAM** ではなく、EEP**ROM** なのか?
2. DRAM を初めて発売し、ひと儲けし…いやかなり儲けた企業はどこか?

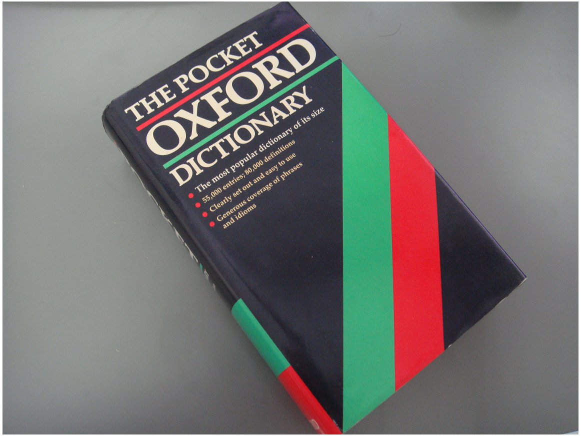
Base n

digital とは?

- digital = digit の形容詞形
- digit って?

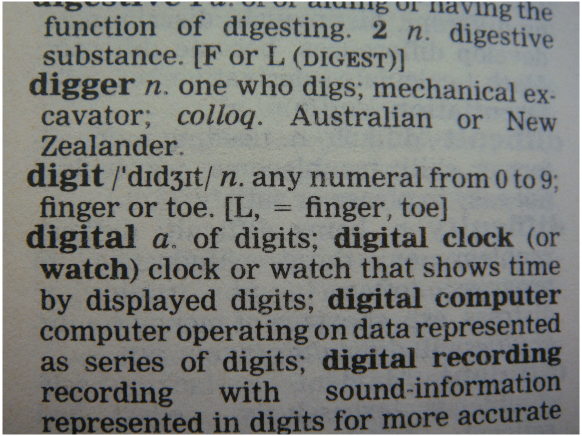
digital とは?

- digital = digit の形容詞形
- digit って?



digital とは?

- digital = digit の形容詞形
- digit って?



function of digesting. **2** *n.* digestive substance. [F or L (DIGEST)]

digger *n.* one who digs; mechanical excavator; *colloq.* Australian or New Zealander.

digit /ˈdɪdʒɪt/ *n.* any numeral from 0 to 9; finger or toe. [L, = finger, toe]

digital *a.* of digits; **digital clock** (or **watch**) clock or watch that shows time by displayed digits; **digital computer** computer operating on data represented as series of digits; **digital recording** recording with sound-information represented in digits for more accurate

BASE n

n 進数: n 種類の を用いて数値を表すやり方

- n 進数の n のことを と言う。
- 10 進数: 0 ∼ 9 の (数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の (数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の (数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, の (数字) を用いて数値を表すやり方

n 進数の表記方法

- カッコで囲んで下つきで基数を書く[a]。省略したら 10 進数。
- 例: $13 = (\_\_\_\_\_)_2 = (\_\_\_\_\_)_8 = (\_\_\_\_\_)_{16}$

[a]カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

BASE n

n 進数: n 種類のディジット(数字)を用いて数値を表すやり方

- n 進数の n のことを と言う。
- 10 進数: 0 ∼ 9 の (数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の (数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の (数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, の (数字) を用いて数値を表すやり方

n 進数の表記方法

- カッコで囲んで下つきで基数を書く[a]。省略したら 10 進数。
- 例: $13 = (\_\_\_\_\_\_)_2 = (\_\_\_\_\_\_)_8 = (\_\_\_\_\_\_)_{16}$

[a]カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

BASE n

n 進数: n 種類のディジット(数字)を用いて数値を表すやり方

- n 進数の n のことを**基数 (base)**と言う。
- 10 進数: 0 ∼ 9 の (数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の (数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の (数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, の (数字) を用いて数値を表すやり方

n 進数の表記方法

- カッコで囲んで下つきで基数を書く[a]。省略したら 10 進数。
- 例: $13 = (\_\_\_\_\_\_)_2 = (\_\_\_\_\_\_)_8 = (\_\_\_\_\_\_)_{16}$

[a]カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

BASE n

n 進数: n 種類の**ディジット**(数字)を用いて数値を表すやり方

- n 進数の n のことを**基数 (base)**と言う。
- 10 進数: 0 ∼ 9 の**ディジット**(数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の**ディジット**(数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の**ディジット**(数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, のディジット(数字) を用いて数値を表すやり方

n 進数の表記方法

- カッコで囲んで下つきで基数を書く[a]。省略したら 10 進数。
- 例: $13 = (\_\_\_\_\_)_2 = (\_\_\_\_\_)_8 = (\_\_\_\_\_)_{16}$

[a] カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

BASE n

n 進数: n 種類の**ディジット**(数字)を用いて数値を表すやり方

- n 進数の n のことを**基数 (base)**と言う。
- 10 進数: 0 ∼ 9 の**ディジット**(数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の**ディジット**(数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の**ディジット**(数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, a, b, c, d, e, fの**ディジット**(数字) を用いて数値を表すやり方

n 進数の表記方法

- カッコで囲んで下つきで基数を書く[a]。省略したら 10 進数。
- 例: $13 = (\_\_\_\_\_\_)_2 = (\_\_\_\_\_\_)_8 = (\_\_\_\_\_\_)_{16}$

[a]カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

BASE n

n 進数: n 種類の**ディジット**(数字)を用いて数値を表すやり方

- n 進数の n のことを**基数 (base)**と言う。
- 10 進数: 0 ∼ 9 の**ディジット**(数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の**ディジット**(数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の**ディジット**(数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, a, b, c, d, e, fの**ディジット**(数字) を用いて数値を表すやり方

n 進数の表記方法

- カッコで囲んで下つきで基数を書く [a]。省略したら 10 進数。
- 例: $13 = (\underline{\ 1101\ })_2 = (\underline{\qquad\quad})_8 = (\underline{\qquad\quad})_{16}$

[a] カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

BASE n

n 進数: n 種類の**ディジット**(数字)を用いて数値を表すやり方

- n 進数の n のことを**基数 (base)**と言う。
- 10 進数: 0 ∼ 9 の**ディジット**(数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の**ディジット**(数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の**ディジット**(数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, a, b, c, d, e, fの**ディジット**(数字) を用いて数値を表すやり方

n 進数の表記方法

- カッコで囲んで下つきで基数を書く[a]。省略したら 10 進数。
- 例: $13 = (\underline{\ 1101\ })_2 = (\underline{\ \ 15\ \ })_8 = (\underline{\qquad})_{16}$

[a]カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

BASE n

n 進数: n 種類の**ディジット**(数字)を用いて数値を表すやり方

- n 進数の n のことを**基数 (base)**と言う。
- 10 進数: 0 ∼ 9 の**ディジット**(数字) を用いて数値を表すやり方
- 2 進数: 0, 1 の**ディジット**(数字) を用いて数値を表すやり方
- 8 進数: 0 ∼ 7 の**ディジット**(数字) を用いて数値を表すやり方
- 16 進数: 0 ∼ 9, a, b, c, d, e, fの**ディジット**(数字) を用いて数値を表すやり方

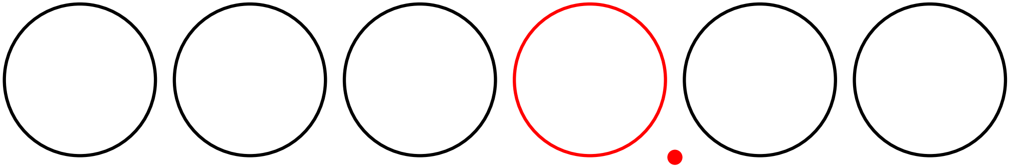
n 進数の表記方法

- カッコで囲んで下つきで基数を書く[a]。省略したら 10 進数。
- 例: $13 = (\underline{\ 1101\ })_2 = (\underline{\ \ 15\ \ })_8 = (\underline{\ \ d\ \ })_{16}$

[a]カッコをつけない場合もあるが、この資料では（明確にするため）カッコをつける。

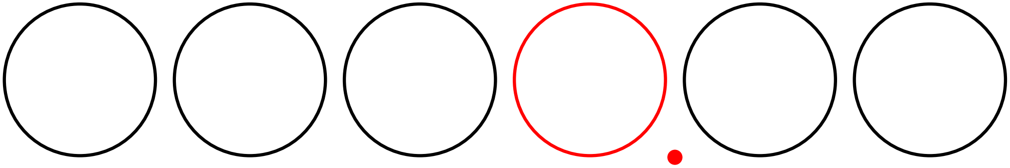
n 進数の書き方・考え方

何のことはない、**左に 1 桁ずれれば** ______ 、**右に 1 桁ずれれば** ______ になるだけ。



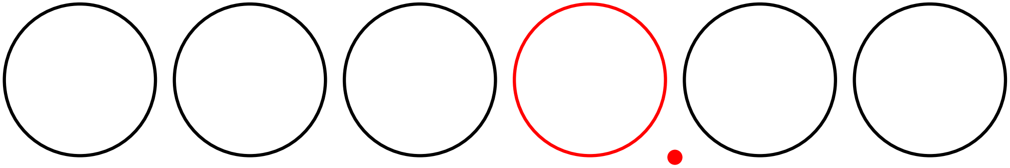
n 進数の書き方・考え方

何のことはない、**左に 1 桁ずれればn 倍**、**右に 1 桁ずれれば** ______ になるだけ。



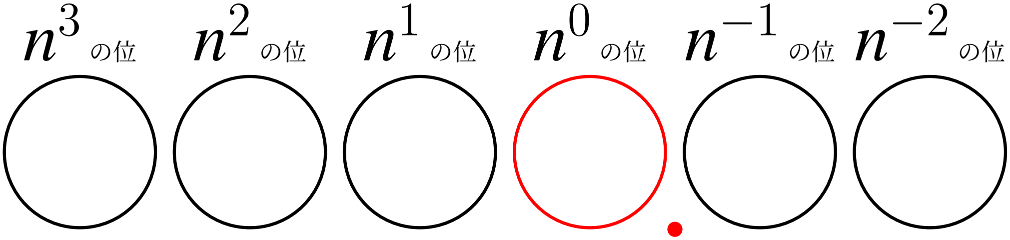
n 進数の書き方・考え方

何のことはない、**左に 1 桁ずれればn 倍**、**右に 1 桁ずれれば$1/n$ 倍**になるだけ。



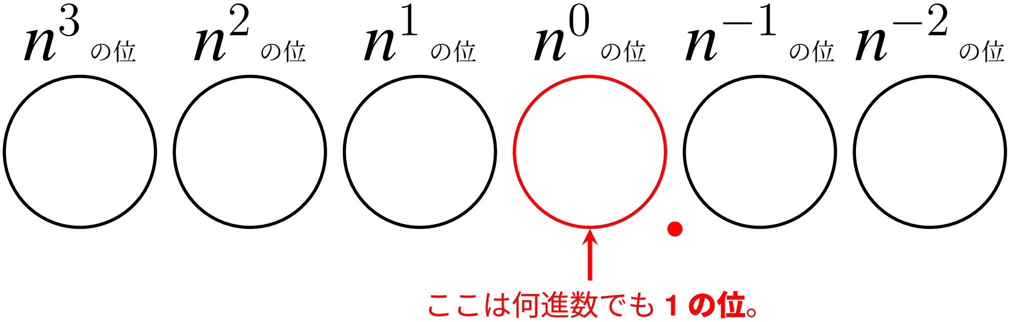
n 進数の書き方・考え方

何のことはない、**左に 1 桁ずれれば n 倍**、**右に 1 桁ずれれば $1/n$ 倍**になるだけ。



n 進数の書き方・考え方

何のことはない、**左に 1 桁ずれればn 倍**、**右に 1 桁ずれれば$1/n$ 倍**になるだけ。



練習: n 進数から 10 進数へ

前のページのとおりにやれば簡単！

問. 10 進数に変換せよ。

1. $(123)_4 = (\qquad)_{10}$
2. $(23.4)_5 = (\qquad\qquad)_{10}$
3. $(1101.011)_2 = (\qquad\qquad\qquad)_{10}$

練習: n 進数から 10 進数へ

前のページのとおりにやれば簡単！

問. 10 進数に変換せよ。

1. $(123)_4 = (\ 27\ )_{10}$
2. $(23.4)_5 = (\qquad\qquad)_{10}$
3. $(1101.011)_2 = (\qquad\qquad\qquad)_{10}$

練習: n 進数から 10 進数へ

前のページのとおりにやれば簡単！

問. 10 進数に変換せよ。

1. $(123)_4 = (\ 27\ )_{10}$
2. $(23.4)_5 = (\ 13.8\ )_{10}$
3. $(1101.011)_2 = (\qquad\qquad)_{10}$

練習: n 進数から 10 進数へ

前のページのとおりにやれば簡単！

問. 10 進数に変換せよ。

1. $(123)_4 = (\ 27\ )_{10}$
2. $(23.4)_5 = (\ 13.8\ )_{10}$
3. $(1101.011)_2 = (\ 13.375\ )_{10}$

ある数 x を n 進数で表す方法

要するに上位の桁から n 進数の各ディジットが何かを決めていくだけ

Step 1. 初期化 (最上位の桁を求める。)

$n^i \leq x$ なる最大の i を求める。($i < 0$ の場合は $i = 0$ とする。)

Step 2. メインループ

- x を n^i で割った商（整数）を p, 余りを q とする。
- 答の i 桁目が p で確定。
- $x \leftarrow q$
- $i \leftarrow i - 1$
- $q = 0$ なら終了。そうでなければ Step 2 を繰り返す。

Ruby で

無駄の多いコードですが、前ページの手順と 1:1 対応に近くしました。

```
# step 1.
n = 4    # 2〜10 進数しか対応してません。
x = 123.453125
i = [(Math.log(x, n)).to_i, 0].max   # なぜ対数だかわかりますよね?
# step 2.
while true
  p = (x / n ** i).to_i
  q = x - (n ** i) * p
  print p.to_s           # 確定した桁を出力
  print "." if i == 0    # 小数点
  x = q
  i = i - 1
  break if q == 0
end
```

練習: 10 進数から n 進数へ

問. 指定の基数の表記に変換せよ。

1. $(9.875)_{10} = (\qquad\qquad\qquad)_2 = (\qquad\quad)_8$
2. $(10.304)_{10} = (\qquad\qquad\quad)_5$

練習: 10 進数から n 進数へ

問. 指定の基数の表記に変換せよ。

1. $(9.875)_{10} = (1001.111)_2 = (\qquad)_8$
2. $(10.304)_{10} = (\qquad\qquad)_5$

練習: 10 進数から n 進数へ

問. 指定の基数の表記に変換せよ。

1. $(9.875)_{10} = (1001.111)_2 = (11.7)_8$
2. $(10.304)_{10} = (\qquad\qquad)_5$

練習: 10 進数から n 進数へ

問. 指定の基数の表記に変換せよ。

1. $(9.875)_{10} = (1001.111)_2 = (11.7)_8$
2. $(10.304)_{10} = (20.123)_5$

2 進数の話

用語

ビット 1 桁のこと。整数の場合 LSB を第 0 ビットとし、MSB に向かって第 1, 2, ⋯ と数える。

MSB ＿＿＿＿＿＿＿＿＿＿。要するに一番　　　のビット。

LSB ＿＿＿＿＿＿＿＿＿＿。要するに一番　　　のビット。

クイズ

MSB, LSB, 第 5 ビットはどれか?

$$(11010111)_2$$

2 進数の話

用語

ビット 1 桁のこと。整数の場合 LSB を第 0 ビットとし、MSB に向かって第 1, 2, $\cdots$ と数える。

MSB ________ Significant Bit。要するに一番 ____ のビット。

LSB ________ Significant Bit。要するに一番 ____ のビット。

クイズ

MSB, LSB, 第 5 ビットはどれか?

$$(11010111)_2$$

2 進数の話

用語

ビット 1 桁のこと。整数の場合 LSB を第 0 ビットとし、MSB に向かって第 1, 2, $\cdots$ と数える。

MSB <u>Most Significant Bit</u>。要するに一番　　　のビット。

LSB <u>　　　　Significant Bit</u>。要するに一番　　　のビット。

クイズ

MSB, LSB, 第 5 ビットはどれか?

$$(11010111)_2$$

2 進数の話

用語

ビット 1 桁のこと。整数の場合 LSB を第 0 ビットとし、MSB に向かって第 1, 2, ⋯ と数える。

MSB Most Significant Bit。要するに一番 左側のビット。

LSB Significant Bit。要するに一番 のビット。

クイズ

MSB, LSB, 第 5 ビットはどれか?

$$(11010111)_2$$

2 進数の話

用語

ビット 1 桁のこと。整数の場合 LSB を第 0 ビットとし、MSB に向かって第 1, 2, $\cdots$ と数える。

MSB Most Significant Bit。要するに一番 左側のビット。

LSB Least Significant Bit。要するに一番　　　のビット。

クイズ

MSB, LSB, 第 5 ビットはどれか?

$$(11010111)_2$$

2 進数の話

用語

ビット 1 桁のこと。整数の場合 LSB を第 0 ビットとし、MSB に向かって第 1, 2, ⋯ と数える。

MSB Most Significant Bit。要するに一番 左側のビット。

LSB Least Significant Bit。要するに一番 右側のビット。

クイズ

MSB, LSB, 第 5 ビットはどれか?

$$(11010111)_2$$

補数 (complement)

compliment(お世辞) とは違います。

補数とは

- ある数の「足すと【　　　　　　　　　】になる相方」のこと。
- **【ちょうどぴったり】**とは 10, 100, 100000 のようなもの、もしくは（桁があふれるのを嫌って）9, 99, 99999 のようなもの（10 進数の場合）。
  - 100000 のようなぴったりを n 進数の場合、　　　　　　と言う。(10 進数なら 10 の補数。)
  - 99999 のようなぴったりを n 進数の場合、　　　　　　と言う。(10 進数なら 9 の補数。)

補数 (complement)

compliment(お世辞) とは違います。

補数とは

- ある数の「足すと**【ちょうどぴったり】**になる相方」のこと。
- **【ちょうどぴったり】**とは 10, 100, 100000 のようなもの、もしくは（桁があふれるのを嫌って）9, 99, 99999 のようなもの（10 進数の場合）。
  - 100000 のようなぴったりを n 進数の場合、 と言う。(10 進数なら 10 の補数。)
  - 99999 のようなぴったりを n 進数の場合、 と言う。(10 進数なら 9 の補数。)

補数 (complement)

compliment(お世辞) とは違います。

補数とは

- ある数の「足すと**【ちょうどぴったり】**になる相方」のこと。
- **【ちょうどぴったり】**とは 10, 100, 100000 のようなもの、もしくは（桁があふれるのを嫌って）9, 99, 99999 のようなもの（10 進数の場合）。
  - 100000 のようなぴったりを n 進数の場合、***n* の補数**と言う。(10 進数なら 10 の補数。)
  - 99999 のようなぴったりを n 進数の場合、　　　　　　　と言う。(10 進数なら 9 の補数。)

補数 (complement)

compliment(お世辞) とは違います。

補数とは

- ある数の「足すと**【ちょうどぴったり】**になる相方」のこと。
- **【ちょうどぴったり】**とは 10, 100, 100000 のようなもの、もしくは（桁があふれるのを嫌って）9, 99, 99999 のようなもの（10 進数の場合）。
  - 100000 のようなぴったりを n 進数の場合、**n の補数**と言う。(10 進数なら 10 の補数。）
  - 99999 のようなぴったりを n 進数の場合、**$n-1$ の補数**と言う。(10 進数なら 9 の補数。）

補数 (complement)

補数の練習問題

- 123 の**9 の補数** =
- 123 の**10 の補数** =
- $(123)_8$ の**7 の補数** =
- $(123)_8$ の**8 の補数** =
- $(10010111)_2$ の**1 の補数** =
- $(10010111)_2$ の**2 の補数** =

- n 進数では『　　の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =　　』
- 2 進数では『1 の補数』は**単に　　!!**
- 2 進数では『2 の補数 =　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** =
- $(123)_8$ の**7 の補数** =
- $(123)_8$ の**8 の補数** =
- $(10010111)_2$ の**1 の補数** =
- $(10010111)_2$ の**2 の補数** =

- n 進数では『　　　の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =　　　　　　　　』
- 2 進数では『1 の補数』は**単に**　　　　　**!!**
- 2 進数では『2 の補数 =　　　　　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** =
- $(123)_8$ の**8 の補数** =
- $(10010111)_2$ の**1 の補数** =
- $(10010111)_2$ の**2 の補数** =

- n 進数では『　　の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =　　』
- 2 進数では『1 の補数』は**単に　　!!**
- 2 進数では『2 の補数 =　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** =
- $(10010111)_2$ の**1 の補数** =
- $(10010111)_2$ の**2 の補数** =

- n 進数では『　　の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 = 　　』
- 2 進数では『1 の補数』は**単に　　!!**
- 2 進数では『2 の補数 = 　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** = $(655)_8$
- $(10010111)_2$ の**1 の補数** =
- $(10010111)_2$ の**2 の補数** =

- n 進数では『　　の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 = 　　　　』
- 2 進数では『1 の補数』は**単に　　　　!!**
- 2 進数では『2 の補数 = 　　　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** = $(655)_8$
- $(10010111)_2$ の**1 の補数** = $(01101000)_2$
- $(10010111)_2$ の**2 の補数** =

- n 進数では『　　　の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =　　　　　　　　』
- 2 進数では『1 の補数』は**単に　　　　　!!**
- 2 進数では『2 の補数 =　　　　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** = $(655)_8$
- $(10010111)_2$ の**1 の補数** = $(01101000)_2$
- $(10010111)_2$ の**2 の補数** = $(01101001)_2$

- n 進数では『　　　の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =　　　　　　　　』
- 2 進数では『1 の補数』は**単に　　　　　!!**
- 2 進数では『2 の補数 =　　　　　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** = $(655)_8$
- $(10010111)_2$ の**1 の補数** = $(01101000)_2$
- $(10010111)_2$ の**2 の補数** = $(01101001)_2$

- n 進数では『$n-1$の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 = 』
- 2 進数では『1 の補数』は**単に !!**
- 2 進数では『2 の補数 = 』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** = $(655)_8$
- $(10010111)_2$ の**1 の補数** = $(01101000)_2$
- $(10010111)_2$ の**2 の補数** = $(01101001)_2$

- n 進数では『$n-1$の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =($n-1$ の補数)+1』
- 2 進数では『1 の補数』は**単に**　　　　　**!!**
- 2 進数では『2 の補数 =　　　　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** = $(655)_8$
- $(10010111)_2$ の**1 の補数** = $(01101000)_2$
- $(10010111)_2$ の**2 の補数** = $(01101001)_2$

- n 進数では『$n-1$の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =($n-1$ の補数)+1』
- 2 進数では『1 の補数』は**単にビット反転!!**
- 2 進数では『2 の補数 =　　　　　　　』

補数 (complement)

補数の練習問題

- 123 の**9 の補数** = 876
- 123 の**10 の補数** = 877
- $(123)_8$ の**7 の補数** = $(654)_8$
- $(123)_8$ の**8 の補数** = $(655)_8$
- $(10010111)_2$ の**1 の補数** = $(01101000)_2$
- $(10010111)_2$ の**2 の補数** = $(01101001)_2$

- n 進数では『$n-1$の補数』を求めるのは**比較的楽**
- n 進数では『n の補数 =($n-1$ の補数)+1』
- 2 進数では『1 の補数』は**単にビット反転!!**
- 2 進数では『2 の補数 =1 の補数 +1』

2 進数による負の表現 (1/3)

符号つき絶対値表現

- ____ を符号ビット $(0 = +,\ 1 = -)$ とし、それ以外のビットを絶対値とするやり方。
- 長所: 単純でわかりやすい（人間にとって）。
- 短所: 機械にとって扱いづらい, 1000 も 0000 も同じ値を表す
- 短所が大きすぎるので**ほとんど使われることはない**。

練習

- 4 ビットの符号つき絶対値表現で表せる値の範囲は ___ 〜 ___ である。
- -5 を 4 ビットの符号つき絶対値表現で表すと ______ である。

2 進数による負の表現 (1/3)

符号つき絶対値表現

- MSB を符号ビット ($0 = +$, $1 = -$) とし、それ以外のビットを絶対値とするやり方。
- 長所: 単純でわかりやすい（人間にとって)。
- 短所: 機械にとって扱いづらい, 1000 も 0000 も同じ値を表す
- 短所が大きすぎるので**ほとんど使われることはない**。

練習

- 4 ビットの符号つき絶対値表現で表せる値の範囲は ____ ～ ____ である。
- -5 を 4 ビットの符号つき絶対値表現で表すと ________ である。

2 進数による負の表現 (1/3)

符号つき絶対値表現

- MSB を符号ビット ($0 = +$, $1 = -$) とし、それ以外のビットを絶対値とするやり方。
- 長所: 単純でわかりやすい（人間にとって）。
- 短所: 機械にとって扱いづらい, 1000 も 0000 も同じ値を表す
- 短所が大きすぎるので**ほとんど使われることはない**。

練習

- 4 ビットの符号つき絶対値表現で表せる値の範囲は -7〜$+7$ である。
- -5 を 4 ビットの符号つき絶対値表現で表すと ________ である。

2 進数による負の表現 (1/3)

符号つき絶対値表現

- MSB を符号ビット ($0 = +$, $1 = -$) とし、それ以外のビットを絶対値とするやり方。
- 長所: 単純でわかりやすい（人間にとって)。
- 短所: 機械にとって扱いづらい, 1000 も 0000 も同じ値を表す
- 短所が大きすぎるので**ほとんど使われることはない**。

練習

- 4 ビットの符号つき絶対値表現で表せる値の範囲は -7〜$+7$ である。
- -5 を 4 ビットの符号つき絶対値表現で表すと $(1101)_2$ である。

2 進数による負の表現 (2/3)

オフセットバイナリ

- $00\cdots0$ に対して、あらかじめ適当な『ゲタ』（通常は $11\cdots1$ との中間値である $10\cdots0$）を履かせた値をゼロとする 2 進数。
- 長所: 計算は比較的単純（計算機にとって）, すべて正数として **大小比較が行える**。
- 短所: ゼロが $00\cdots0$ でなく、不自然。
- ときどき使われる場面もある（例: IEEE 754 の指数部）。

練習

- 4 ビットのオフセットバイナリで表せる値の範囲は ＿＿〜＿＿ 。
- -5 を 4 ビットのオフセットバイナリで表すと ＿＿＿＿ 。
- $+5$ を 4 ビットのオフセットバイナリで表すと ＿＿＿＿ 。

2 進数による負の表現 (2/3)

オフセットバイナリ

- $00 \cdots 0$ に対して、あらかじめ適当な『ゲタ』（通常は $11 \cdots 1$ との中間値である $10 \cdots 0$）を履かせた値をゼロとする 2 進数。
- 長所: 計算は比較的単純（計算機にとって）, すべて正数として **大小比較が行える**。
- 短所: ゼロが $00 \cdots 0$ でなく、不自然。
- ときどき使われる場面もある（例: IEEE 754 の指数部）。

練習

- 4 ビットのオフセットバイナリで表せる値の範囲は $\underline{-8 \sim +7}$ 。
- -5 を 4 ビットのオフセットバイナリで表すと ________ 。
- $+5$ を 4 ビットのオフセットバイナリで表すと ________ 。

2 進数による負の表現 (2/3)

オフセットバイナリ

- $00\cdots0$ に対して、あらかじめ適当な『ゲタ』（通常は $11\cdots1$ との中間値である $10\cdots0$）を履かせた値をゼロとする 2 進数。
- 長所: 計算は比較的単純（計算機にとって），すべて正数として **大小比較が行える**。
- 短所: ゼロが $00\cdots0$ でなく、不自然。
- ときどき使われる場面もある（例: IEEE 754 の指数部）。

練習

- 4 ビットのオフセットバイナリで表せる値の範囲は $\underline{-8 \sim +7}$ 。
- -5 を 4 ビットのオフセットバイナリで表すと $\underline{(0011)_2}$ 。
- $+5$ を 4 ビットのオフセットバイナリで表すと ________ 。

2 進数による負の表現 (2/3)

オフセットバイナリ

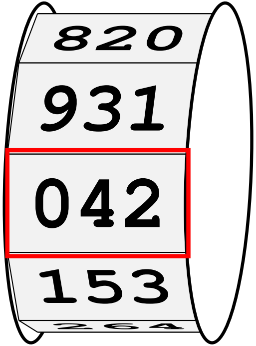
- $00\cdots0$ に対して、あらかじめ適当な『ゲタ』（通常は $11\cdots1$ との中間値である $10\cdots0$）を履かせた値をゼロとする 2 進数。
- 長所: 計算は比較的単純（計算機にとって）, すべて正数として **大小比較が行える**。
- 短所: ゼロが $00\cdots0$ でなく、不自然。
- ときどき使われる場面もある（例: IEEE 754 の指数部）。

練習

- 4 ビットのオフセットバイナリで表せる値の範囲は $\underline{-8\sim+7}$ 。
- -5 を 4 ビットのオフセットバイナリで表すと $\underline{(0011)_2}$ 。
- $+5$ を 4 ビットのオフセットバイナリで表すと $\underline{(1101)_2}$ 。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

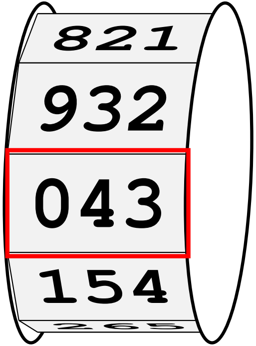


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は___ で表すのが自然。
- -2 は___ で表すのが自然。
- $-n$ は________ で表すのが自然。
- つまり、負数は_________ で表すのが自然。
- n 進数の負数は________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

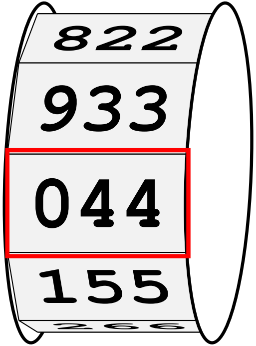


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は___ で表すのが自然。
- -2 は___ で表すのが自然。
- $-n$ は________ で表すのが自然。
- つまり、負数は__________ で表すのが自然。
- n 進数の負数は_________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

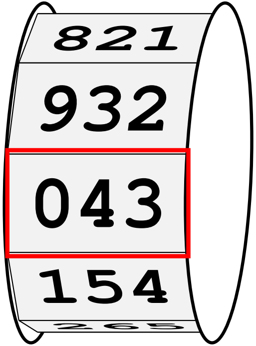


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は___ で表すのが自然。
- -2 は___ で表すのが自然。
- $-n$ は_______ で表すのが自然。
- つまり、負数は________ で表すのが自然。
- n 進数の負数は_______ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

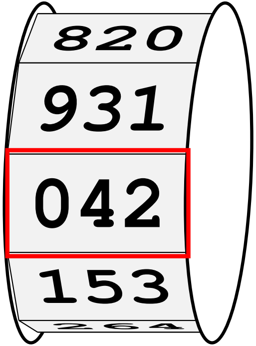


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は___ で表すのが自然。
- -2 は___ で表すのが自然。
- $-n$ は_______ で表すのが自然。
- つまり、負数は________ で表すのが自然。
- n 進数の負数は_______ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

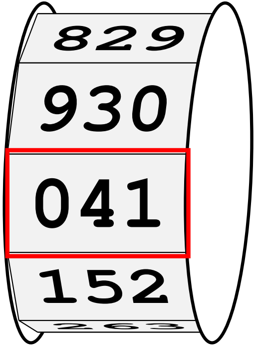


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は___ で表すのが自然。
- -2 は___ で表すのが自然。
- $-n$ は_________ で表すのが自然。
- つまり、負数は__________ で表すのが自然。
- n 進数の負数は_________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

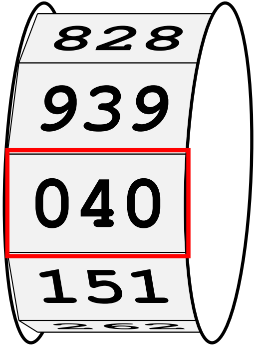


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は___ で表すのが自然。
- -2 は___ で表すのが自然。
- $-n$ は______ で表すのが自然。
- つまり、負数は________ で表すのが自然。
- n 進数の負数は________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

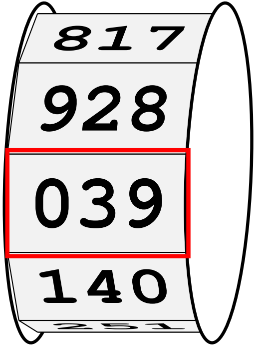


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は___ で表すのが自然。
- -2 は___ で表すのが自然。
- $-n$ は________ で表すのが自然。
- つまり、負数は__________ で表すのが自然。
- n 進数の負数は_________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

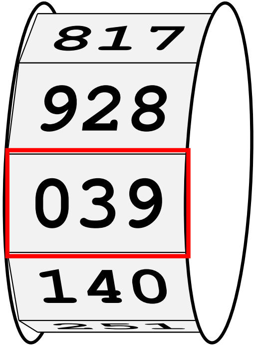


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は____ で表すのが自然。
- -2 は____ で表すのが自然。
- $-n$ は__________ で表すのが自然。
- つまり、負数は____________ で表すのが自然。
- n 進数の負数は__________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

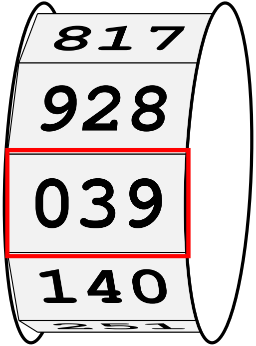


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は999 で表すのが自然。
- -2 は____ で表すのが自然。
- $-n$ は________ で表すのが自然。
- つまり、負数は__________ で表すのが自然。
- n 進数の負数は_________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

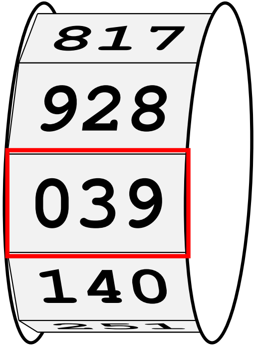


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は999 で表すのが自然。
- -2 は998 で表すのが自然。
- $-n$ は________ で表すのが自然。
- つまり、負数は__________ で表すのが自然。
- n 進数の負数は_________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

よくあるカウンタ

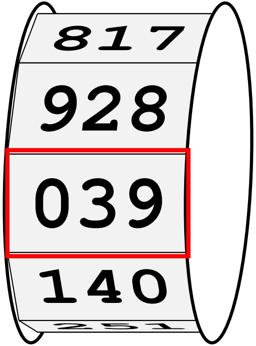


考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は$\underline{999}$ で表すのが自然。
- -2 は$\underline{998}$ で表すのが自然。
- $-n$ は$\underline{1000-n}$ で表すのが自然。
- つまり、負数は__________ で表すのが自然。
- n 進数の負数は__________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。

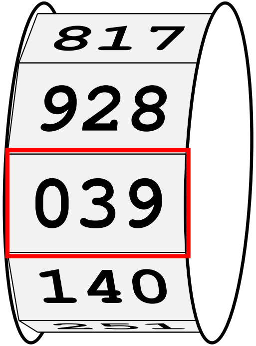
よくあるカウンタ



考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は<u>999</u> で表すのが自然。
- -2 は<u>998</u> で表すのが自然。
- $-n$ は$\underline{1000 - n}$ で表すのが自然。
- つまり、負数は<u>**10 の補数**</u> で表すのが自然。
- n 進数の負数は________ で表すのが自然。

正数縛りで数を数えることについて考えてみる。



考えてみよう

- 10 進数で 0〜999 までカウントできるカウンタで負の数を表現したければどうする?
- -1 は<u>999</u> で表すのが自然。
- -2 は<u>998</u> で表すのが自然。
- $-n$ は$\underline{1000-n}$ で表すのが自然。
- つまり、負数は<u>**10 の補数**</u> で表すのが自然。
- n 進数の負数は<u>**n の補数**</u> で表すのが自然。

2 進数による負の表現 (3/3)

2 の補数表現

- ある数値の 2 の補数をその数の -1 倍とするやり方。
- 長所: 人間にとっても機械にとっても自然, 機械にとって **計算がすごく楽**。
- 短所: 特にない。
- ほとんどのケースで **負を表現する時は 2 の補数が使われる**。

練習

- 4 ビットの 2 の補数表現で表せる値の範囲は ____ ～ ____ 。
- -5 を 4 ビットの 2 の補数表現で表すと ________ 。
- 16 ビットの 2 の補数表現で表せる値の範囲は ________ ～ ________ 。

2 進数による負の表現 (3/3)

2 の補数表現

- ある数値の 2 の補数をその数の −1 倍とするやり方。
- 長所: 人間にとっても機械にとっても自然, 機械にとって **計算がすごく楽**。
- 短所: 特にない。
- ほとんどのケースで**負を表現する時は 2 の補数が使われる**。

練習

- 4 ビットの 2 の補数表現で表せる値の範囲は −8〜+7 。
- −5 を 4 ビットの 2 の補数表現で表すと ＿＿＿＿ 。
- 16 ビットの 2 の補数表現で表せる値の範囲は ＿＿＿＿ 〜 ＿＿＿＿ 。

2 進数による負の表現 (3/3)

2 の補数表現

- ある数値の 2 の補数をその数の -1 倍とするやり方。
- 長所: 人間にとっても機械にとっても自然, 機械にとって **計算がすごく楽**。
- 短所: 特にない。
- ほとんどのケースで**負を表現する時は 2 の補数が使われる**。

練習

- 4 ビットの 2 の補数表現で表せる値の範囲は $\underline{-8\sim+7}$ 。
- -5 を 4 ビットの 2 の補数表現で表すと $\underline{(1011)_2}$ 。
- 16 ビットの 2 の補数表現で表せる値の範囲は $\underline{\qquad\quad\sim\qquad\quad}$ 。

2 進数による負の表現 (3/3)

2 の補数表現

- ある数値の 2 の補数をその数の -1 倍とするやり方。
- 長所: 人間にとっても機械にとっても自然, 機械にとって **計算がすごく楽**。
- 短所: 特にない。
- ほとんどのケースで**負を表現する時は 2 の補数が使われる**。

練習

- 4 ビットの 2 の補数表現で表せる値の範囲は $\underline{-8 \sim +7}$ 。
- -5 を 4 ビットの 2 の補数表現で表すと $\underline{(1011)_2}$ 。
- 16 ビットの 2 の補数表現で表せる値の範囲は $\underline{-32768 \sim +32767}$ 。

2 進数による負の表現のまとめ練習

Q: 以下の表を完成させよ。ただし二進数は 3 ビットとし、カッコと下つき数字は省略して良い。

値	符号つき絶対値	オフセットバイナリ	2 の補数表現
-4	N/A	000	
-3			
-2			
-1			
0			
$+1$			
$+2$			
$+3$			

2 進数による負の表現のまとめ練習

Q: 以下の表を完成させよ。ただし二進数は 3 ビットとし、カッコと下つき数字は省略して良い。

値	符号つき絶対値	オフセットバイナリ	2 の補数表現
-4	N/A	000	
-3	111		
-2	110		
-1	101		
0	000 or 100		
$+1$	001		
$+2$	010		
$+3$	011		

2 進数による負の表現のまとめ練習

Q: 以下の表を完成させよ。ただし二進数は 3 ビットとし、カッコと下つき数字は省略して良い。

値	符号つき絶対値	オフセットバイナリ	2 の補数表現
-4	N/A	000	
-3	111	001	
-2	110	010	
-1	101	011	
0	000 or 100	100	
$+1$	001	101	
$+2$	010	110	
$+3$	011	111	

2 進数による負の表現のまとめ練習

Q: 以下の表を完成させよ。ただし二進数は 3 ビットとし、カッコと下つき数字は省略して良い。

値	符号つき絶対値	オフセットバイナリ	2 の補数表現
-4	N/A	000	100
-3	111	001	101
-2	110	010	110
-1	101	011	111
0	000 or 100	100	000
$+1$	001	101	001
$+2$	010	110	010
$+3$	011	111	011

符号拡張

符号拡張 (sign extension)

2 の補数表現された 2 進数のビット数を増やすこと。MSB を好きなだけ複製しても『数値』は変わらない。

Q: 次の 10 進数の数値を 4 ビットの 2 の補数表現し、さらに 6 ビットに符号拡張せよ。

- $(5)_{10} =$
- $(-3)_{10} =$
- $(-8)_{10} =$

符号拡張

符号拡張 (sign extension)

2 の補数表現された 2 進数のビット数を増やすこと。MSB を好きなだけ複製しても『数値』は変わらない。

Q: 次の 10 進数の数値を 4 ビットの 2 の補数表現し、さらに 6 ビットに符号拡張せよ。

- $(5)_{10} = (0101)_2 =$
- $(-3)_{10} =$
- $(-8)_{10} =$

符号拡張

符号拡張 (sign extension)

2 の補数表現された 2 進数のビット数を増やすこと。MSB を好きなだけ複製しても『数値』は変わらない。

Q: 次の 10 進数の数値を 4 ビットの 2 の補数表現し、さらに 6 ビットに符号拡張せよ。

- $(5)_{10} = (0101)_2 = (000101)_2$
- $(-3)_{10} =$
- $(-8)_{10} =$

符号拡張

符号拡張 (sign extension)

2 の補数表現された 2 進数のビット数を増やすこと。MSB を好きなだけ複製しても『数値』は変わらない。

Q: 次の 10 進数の数値を 4 ビットの 2 の補数表現し、さらに 6 ビットに符号拡張せよ。

- $(5)_{10} = (0101)_2 = (000101)_2$
- $(-3)_{10} = (1101)_2 =$
- $(-8)_{10} =$

符号拡張

符号拡張 (sign extension)

2 の補数表現された 2 進数のビット数を増やすこと。MSB を好きなだけ複製しても『数値』は変わらない。

Q: 次の 10 進数の数値を 4 ビットの 2 の補数表現し、さらに 6 ビットに符号拡張せよ。

- $(5)_{10} = (0101)_2 = (000101)_2$
- $(-3)_{10} = (1101)_2 = (111101)_2$
- $(-8)_{10} =$

符号拡張

符号拡張 (sign extension)

2 の補数表現された 2 進数のビット数を増やすこと。MSB を好きなだけ複製しても『数値』は変わらない。

Q: 次の 10 進数の数値を 4 ビットの 2 の補数表現し、さらに 6 ビットに符号拡張せよ。

- $(5)_{10} = (0101)_2 = (000101)_2$
- $(-3)_{10} = (1101)_2 = (111101)_2$
- $(-8)_{10} = (1000)_2 =$

符号拡張

符号拡張 (sign extension)

2 の補数表現された 2 進数のビット数を増やすこと。MSB を好きなだけ複製しても『数値』は変わらない。

Q: 次の 10 進数の数値を 4 ビットの 2 の補数表現し、さらに 6 ビットに符号拡張せよ。

- $(5)_{10} = (0101)_2 = (000101)_2$
- $(-3)_{10} = (1101)_2 = (111101)_2$
- $(-8)_{10} = (1000)_2 = (111000)_2$

2 の補数表現の 2 進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は　　　する。

Q: 4 ビットの 2 の補数表現した 2 進数で計算せよ。

- $4 + (-8) =$
- $(-3) \times 2 =$
- $(-3) \times (-2) =$

2の補数表現の2進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は**無視**する。

Q: 4ビットの2の補数表現した2進数で計算せよ。

- $4 + (-8) =$
- $(-3) \times 2 =$
- $(-3) \times (-2) =$

2の補数表現の2進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は**無視**する。

Q: 4ビットの2の補数表現した2進数で計算せよ。

- $4 + (-8) = (0100)_2 + (1000)_2 =$
- $(-3) \times 2 =$
- $(-3) \times (-2) =$

2 の補数表現の 2 進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は**無視**する。

Q: 4 ビットの 2 の補数表現した 2 進数で計算せよ。

- $4 + (-8) = (0100)_2 + (1000)_2 = (1100)_2$
- $(-3) \times 2 =$
- $(-3) \times (-2) =$

2 の補数表現の 2 進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は**無視**する。

Q: 4 ビットの 2 の補数表現した 2 進数で計算せよ。

- $4 + (-8) = (0100)_2 + (1000)_2 = (1100)_2$
- $(-3) \times 2 = (1101)_2 \times (0010)_2 =$
- $(-3) \times (-2) =$

2 の補数表現の 2 進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は**無視**する。

Q: 4 ビットの 2 の補数表現した 2 進数で計算せよ。

- $4 + (-8) = (0100)_2 + (1000)_2 = (1100)_2$
- $(-3) \times 2 = (1101)_2 \times (0010)_2 = (1010)_2$
- $(-3) \times (-2) =$

2 の補数表現の 2 進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は**無視**する。

Q: 4 ビットの 2 の補数表現した 2 進数で計算せよ。

- $4 + (-8) = (0100)_2 + (1000)_2 = (1100)_2$
- $(-3) \times 2 = (1101)_2 \times (0010)_2 = (1010)_2$
- $(-3) \times (-2) =$ (必ず自分でやろう) $= (0110)_2$

2の補数表現の2進数の加算・乗算

case 1: 答えが表現可能範囲（あふれない）の場合

1. そのまま計算する。
2. はみ出る部分は**無視**する。

Q: 4ビットの2の補数表現した2進数で計算せよ。

- $4 + (-8) = (0100)_2 + (1000)_2 = (1100)_2$
- $(-3) \times 2 = (1101)_2 \times (0010)_2 = (1010)_2$
- $(-3) \times (-2) = (\text{必ず自分でやろう}) = (0110)_2$

でも答えが溢れるかどうかなんてやってみないとわからないのでは…?

2 の補数表現の 2 進数の加算・乗算

case 2: 答えが表現可能範囲かどうか不明な場合

1. 答えが取りうる最大のビット数まで、足す数/掛ける数をあらかじめ　　　　する。
   - 加算の場合はもとのビット数 +1。
   - 乗算の場合はもとのビット数の 2 倍。
2. そのまま計算する。
3. はみ出る部分は　　　する。

Q: 次の計算を 2 の補数表現した 2 進数で行え。

- $(-9) + (-8) =$
- $(-3) \times 7 =$
- $(-8) \times (-8) =$

2の補数表現の2進数の加算・乗算

case 2: 答えが表現可能範囲かどうか不明な場合

1. 答えが取りうる最大のビット数まで、足す数/掛ける数をあらかじめ **符号拡張** する。
   - 加算の場合はもとのビット数 +1。
   - 乗算の場合はもとのビット数の 2 倍。
2. そのまま計算する。
3. はみ出る部分は　　する。

Q: 次の計算を2の補数表現した2進数で行え。

- $(-9) + (-8) =$
- $(-3) \times 7 =$
- $(-8) \times (-8) =$

2の補数表現の2進数の加算・乗算

case 2: 答えが表現可能範囲かどうか不明な場合

1. 答えが取りうる最大のビット数まで、足す数/掛ける数をあらかじめ**符号拡張**する。
   - 加算の場合はもとのビット数 +1。
   - 乗算の場合はもとのビット数の 2 倍。
2. そのまま計算する。
3. はみ出る部分は**無視**する。

Q: 次の計算を 2 の補数表現した 2 進数で行え。

- $(-9) + (-8) =$
- $(-3) \times 7 =$
- $(-8) \times (-8) =$

2 の補数表現の 2 進数の加算・乗算

case 2: 答えが表現可能範囲かどうか不明な場合

1. 答えが取りうる最大のビット数まで、足す数/掛ける数をあらかじめ **符号拡張** する。
   - 加算の場合はもとのビット数 +1。
   - 乗算の場合はもとのビット数の 2 倍。
2. そのまま計算する。
3. はみ出る部分は **無視** する。

Q: 次の計算を 2 の補数表現した 2 進数で行え。

- $(-9)+(-8)=(110111)_2+(111000)_2=$
- $(-3)\times 7=$
- $(-8)\times(-8)=$

2 の補数表現の 2 進数の加算・乗算

case 2: 答えが表現可能範囲かどうか不明な場合

1. 答えが取りうる最大のビット数まで、足す数/掛ける数をあらかじめ **符号拡張**する。
   - 加算の場合はもとのビット数 +1。
   - 乗算の場合はもとのビット数の 2 倍。
2. そのまま計算する。
3. はみ出る部分は**無視**する。

Q: 次の計算を 2 の補数表現した 2 進数で行え。

- $(-9) + (-8) = (110111)_2 + (111000)_2 = (101111)_2$
- $(-3) \times 7 =$
- $(-8) \times (-8) =$

2 進数 →2^n 進数変換

2 進数 →8 進数

$$1\ 1\ 0\ 1\ 1\ 0\ 1\ 0.\ 0\ 1\ 1$$

パディング

1. **小数点**を基準に　桁ずつ区切る。(空白は 0 で埋める)
2. 各区切り毎に 2 進数 →8 進数変換する。

※

2 進数 →2^n 進数変換

2 進数 →8 進数

$$1\,1\,0\,1\,1\,0\,1\,0.\,0\,1\,1$$

1. **小数点**を基準に3桁ずつ区切る。(空白は 0で埋める(パディング))
2. 各区切り毎に 2 進数 →8 進数変換する。

※

2 進数 $\rightarrow 2^n$ 進数変換

2 進数 →8 進数

$$\textcolor{gray}{0}\ 1\ 1\,|\,0\ 1\ 1\,|\,0\ 1\ 0.\,|\,0\ 1\ 1$$

1. **小数点**を基準に3桁ずつ区切る。(空白は 0で埋める(パディング))
2. 各区切り毎に 2 進数 →8 進数変換する。

※ $8 = 2^3$ だから 3 桁ずつ区切る。

2 進数 →2^n 進数変換

2 進数 →8 進数

$$
\begin{array}{ccc|ccc|ccc|ccc}
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0. & 0 & 1 & 1 \\
(& 3 & & & 3 & & & 2. & & & 3 &)_8
\end{array}
$$

① **小数点**を基準に3桁ずつ区切る。(空白は 0で埋める(パディング))

② 各区切り毎に 2 進数 →8 進数変換する。

※ $8 = 2^3$ だから 3 桁ずつ区切る。

2 進数 →16 進数

$$1\,1\,0\,1\,1\,0\,1\,0.\,0\,1\,1$$

1. **小数点**を基準に　桁ずつ区切る。(空白は 0 で埋める(パディング))
2. 各区切り毎に 2 進数 →16 進数変換する。

2 進数 →16 進数

$$1\,1\,0\,1\,1\,0\,1\,0.\,0\,1\,1$$

1. **小数点**を基準に4桁ずつ区切る。(空白は 0で埋める(パディング))
2. 各区切り毎に 2 進数 →16 進数変換する。

2 進数 →2^n 進数変換

2 進数 →16 進数

$$1\,1\,0\,1 \,|\, 1\,0\,1\,0.\,|\,0\,1\,1\,0$$

1. **小数点**を基準に4桁ずつ区切る。(空白は 0で埋める(パディング))
2. 各区切り毎に 2 進数 →16 進数変換する。

2 進数 $\rightarrow 2^n$ 進数変換

2 進数 →16 進数

$$\begin{array}{cccc|cccc|cccc} 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0. & 0 & 1 & 1 & 0 \end{array}$$

$$(\;\mathrm{D} \qquad \mathrm{A}. \qquad 6\;)_{16}$$

1. **小数点**を基準に4桁ずつ区切る。(空白は 0で埋める(パディング))
2. 各区切り毎に 2 進数 →16 進数変換する。

2 進数 →2^n 進数変換

2 進数 →16 進数

$$\begin{array}{cccc|cccc|cccc} 1&1&0&1&1&0&1&0.&0&1&1&0 \end{array}$$

$$(\mathrm{D}\quad\mathrm{A}.\quad 6)_{16}$$

1. **小数点**を基準に4桁ずつ区切る。(空白は 0で埋める(パディング))
2. 各区切り毎に 2 進数 →16 進数変換する。

2 進数 →2^n 進数はとても簡単!

n ずつ区切って、それぞれ 2 進数 →n 進数変換する。

2^n 進数 →2 進数変換もとても簡単

2^n 進数 $\rightarrow 2$ 進数変換もとても簡単

一桁毎に 2 進数に変換するだけ！

前ページの例を見れば一目瞭然

出席確認レポート課題 (次の月曜の 12 時締め切り)

問 1(全員): 8 桁の 2 の補数表現の 2 進数で、$(125)_{10}, (-42)_{10}, (-242)_7$ をそれぞれ表わせ。

問 2(できる人): 正の数を表す文字列と基数 (2〜9 の整数) を入力すると 10 進数に変換して出力するプログラムを**何も見ないで、独力で、**作成せよ。言語は問わないので自分が得意な言語、練習したい言語で書いて構わない。解説を添えたソースコードを提出すること。小数点対応は難しかったら省いても良い。

実行例:

```
$ ruby kadai.rb
101010          #<-文字列を入力
2               #<-基数 を入力
42.0            #->変換結果(10進数)を出力
$ ruby kadai.rb # もう一度実行
20.123          #<-文字列を入力
5               #<-基数 を入力
10.304          #->変換結果(10進数)を出力
```

提出は下記 URL の Google Forms。歪んでいない、開いた時に横倒しになっていない、コントラストが読むに耐えうる PDF で提出すること。

https://forms.gle/9ruwtfJg5LQgQNpU7