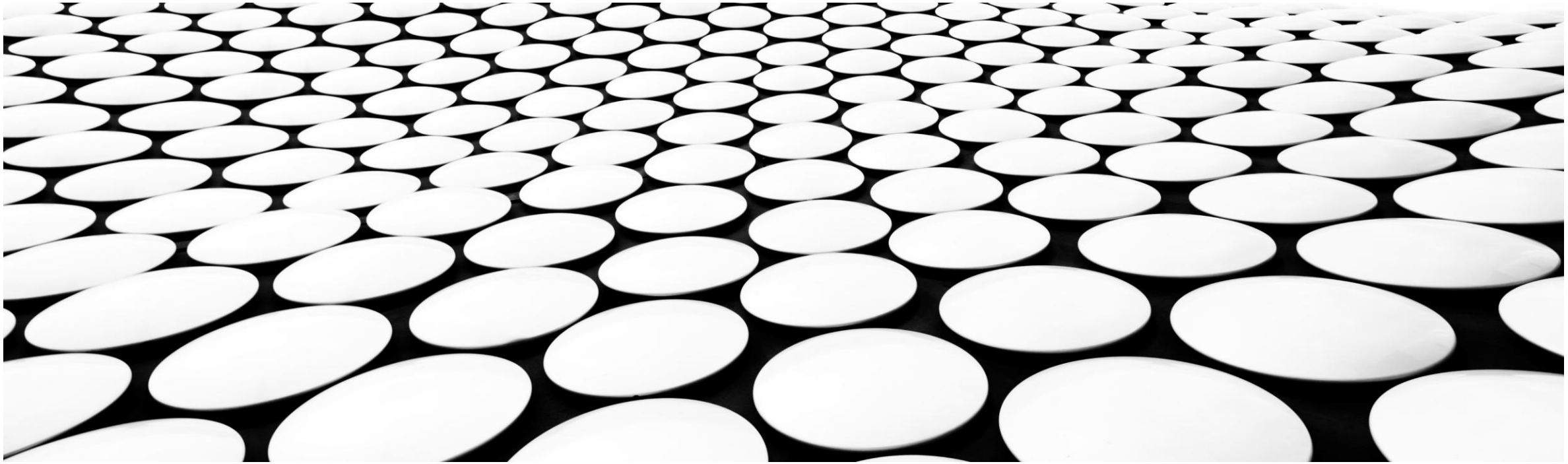
フラッシュメモリ製のオセロコンピュータ

2026/05/09(土)第7回 自作CPUを語る会　自作CPUオセロ大会

にちか (@lxacas)



はじめに

私はこれまで同じテーマで、以下のイベントに登壇させていただきました

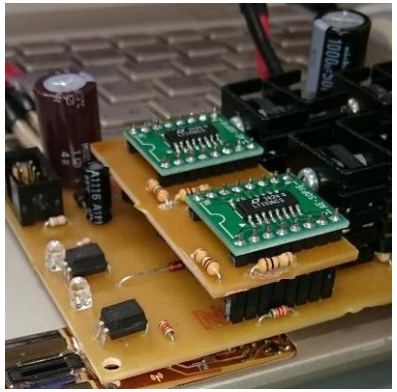
- 第4回 自作CPUを語る会（2024/12/01）
- RECONF（リコンフィギャラブルシステム）研究会（2025/01/28）
- Kernel/VM探検隊 北陸@Part 8（2025/12/06）

なるべく最新の進捗をスライドとして共有いたしますが、

その一部が過去の発表と重複することをご了承ください

自己紹介

- 氏名： にちか
- 職業： 組込みエンジニア
- 趣味： 自作CPU、自作SSD、SSD同人誌の執筆



SSD同人誌1号： 自作SSDで動かす自作CPU

SSD同人誌2号： え！音が鳴るSSDでコンピューティングを！？

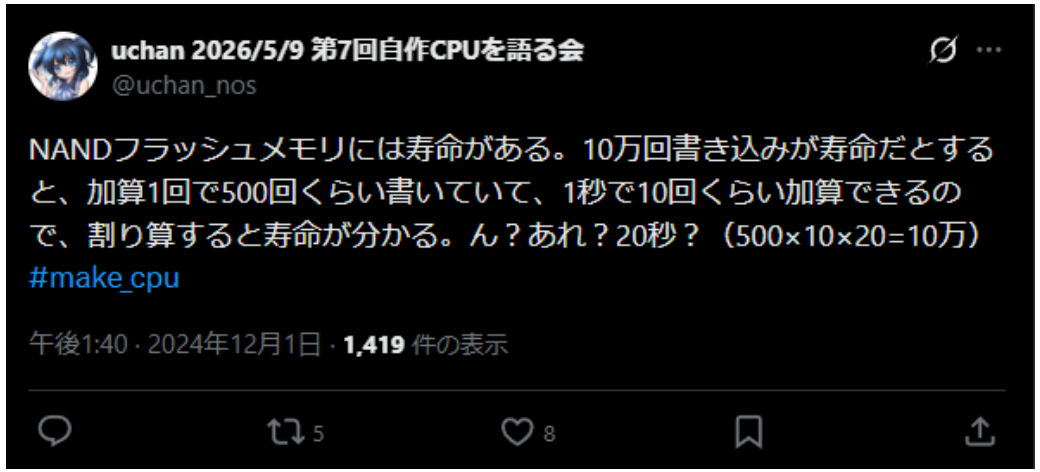
SSD同人誌3号： NAND型フラッシュメモリで創るコンピュータ

SSD同人誌4号： フラッシュメモリの読み書きで作る人工知能

フラッシュメモリはCPUの材料に適してない

意外かもしれませんが

- 読み書き消去操作が遅い（μs、msオーダー）
- 計算するための素子ではない
- 限界まで書き換えると寿命に到達



さらに、

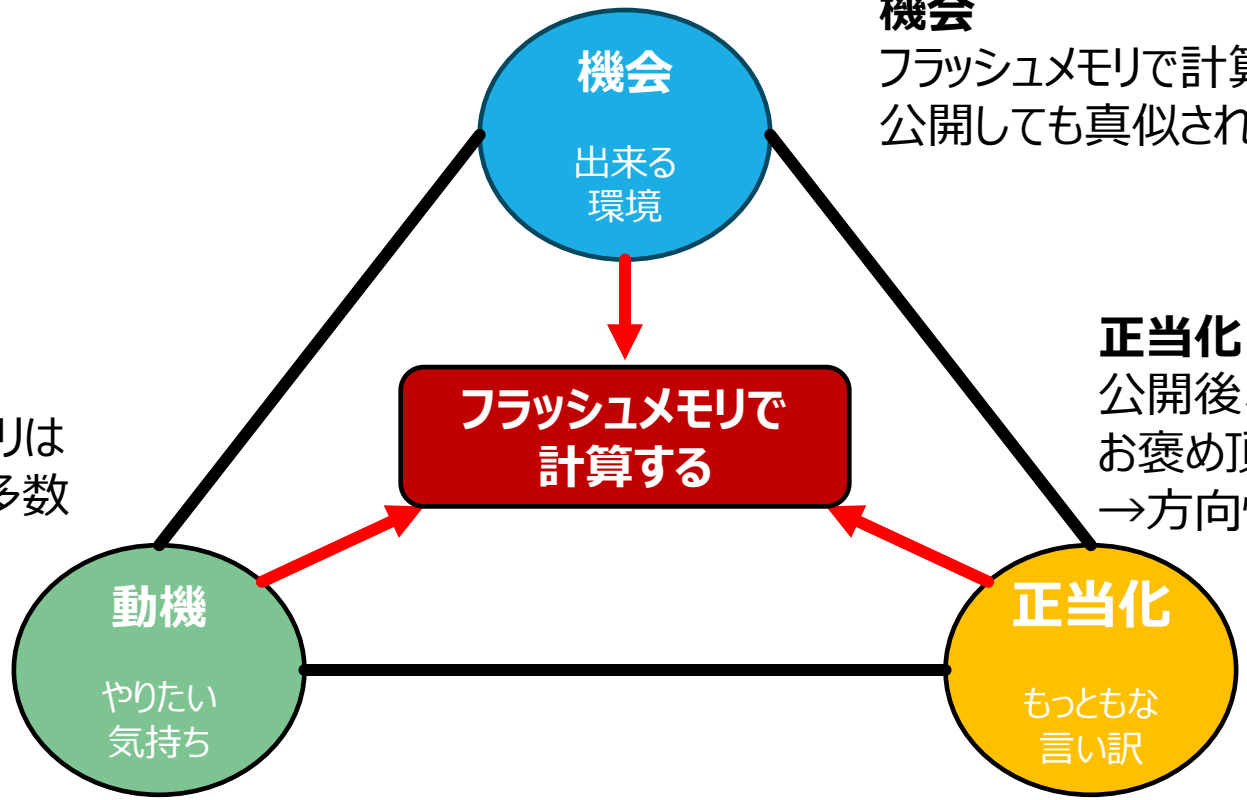
- NAND型フラッシュメモリの価格高騰
- 相次ぐ低容量品のEOL

→計算のために書き潰すだなんてとんでもない！

では何故やってしまうのか

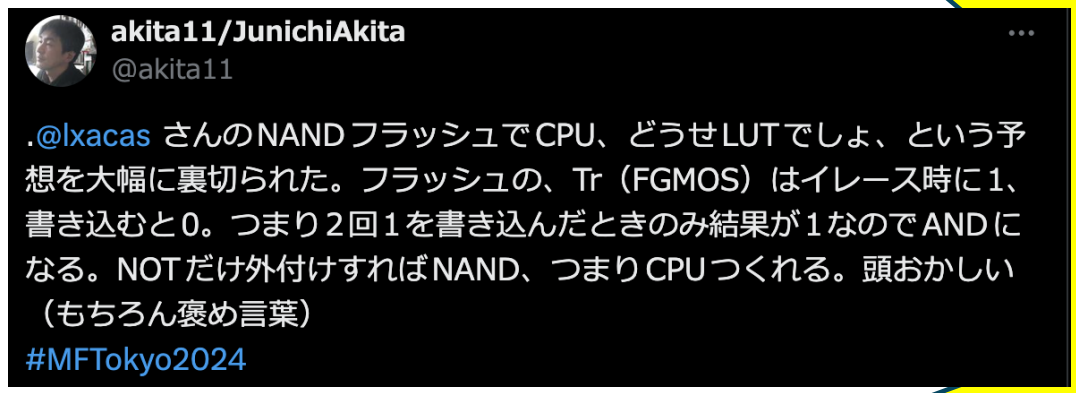
機会
フラッシュメモリで計算が出来ることを誰も知らない
公開しても真似されない

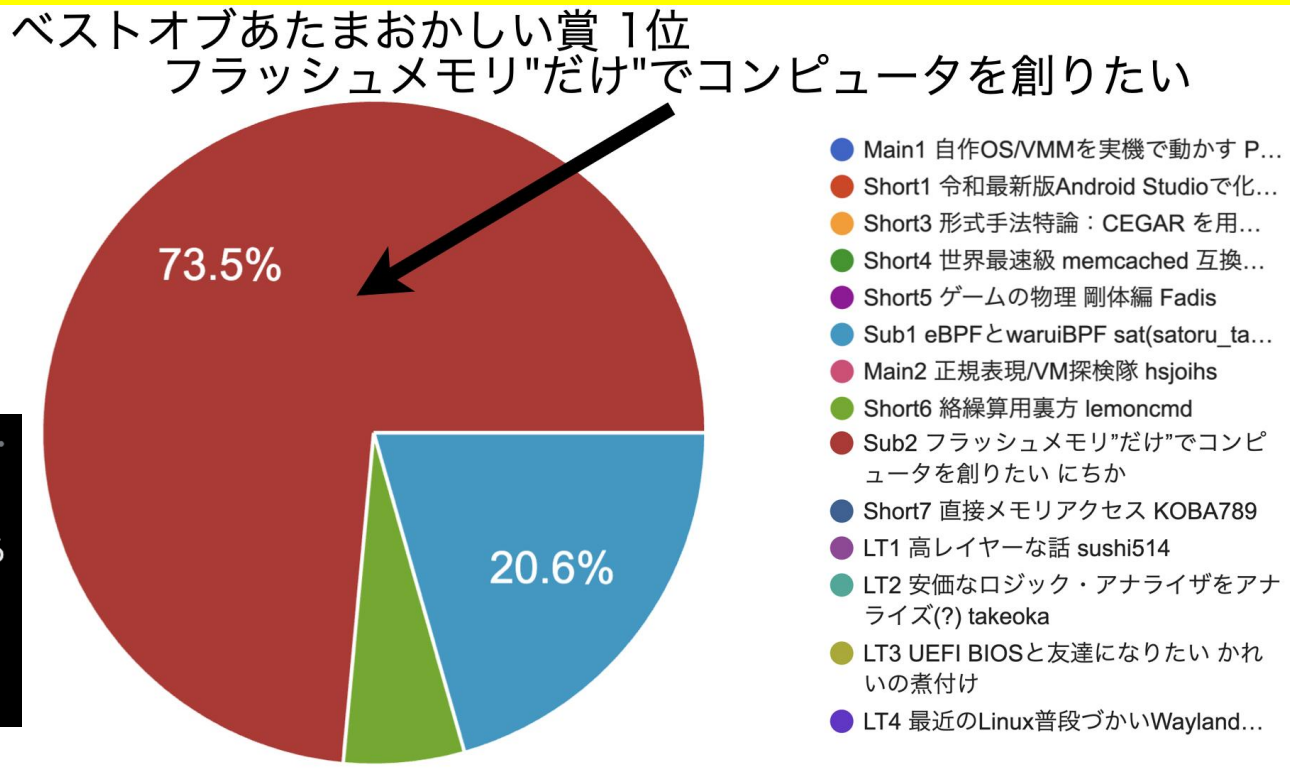
動機
(2023年当時)フラッシュメモリは
演算に無関係と考える人が多数
→計算して見返したい



正当化
公開後、多くの人たちに「頭おかしい」と
お褒め頂く
→方向性は合っていそう

では何故やってしまうのか

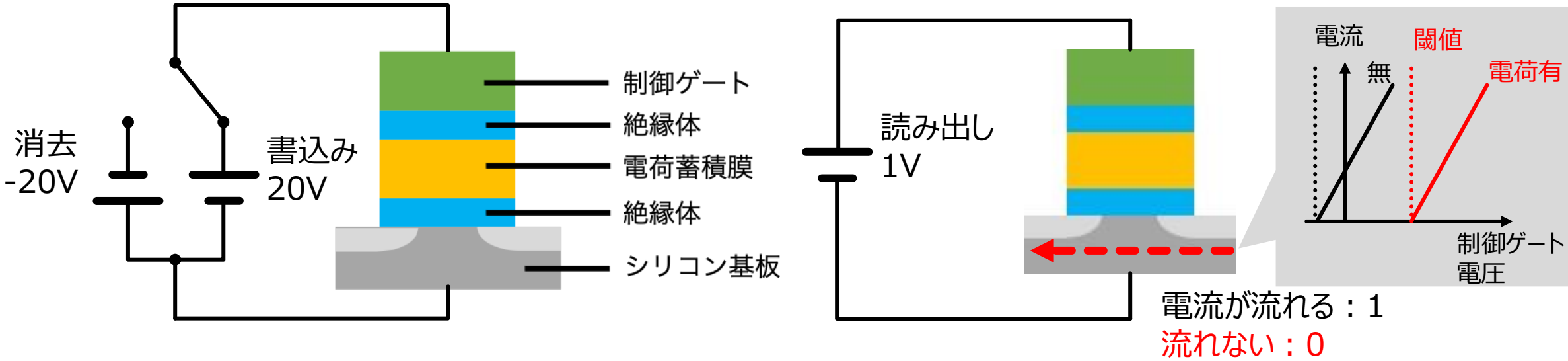




アジェンダ

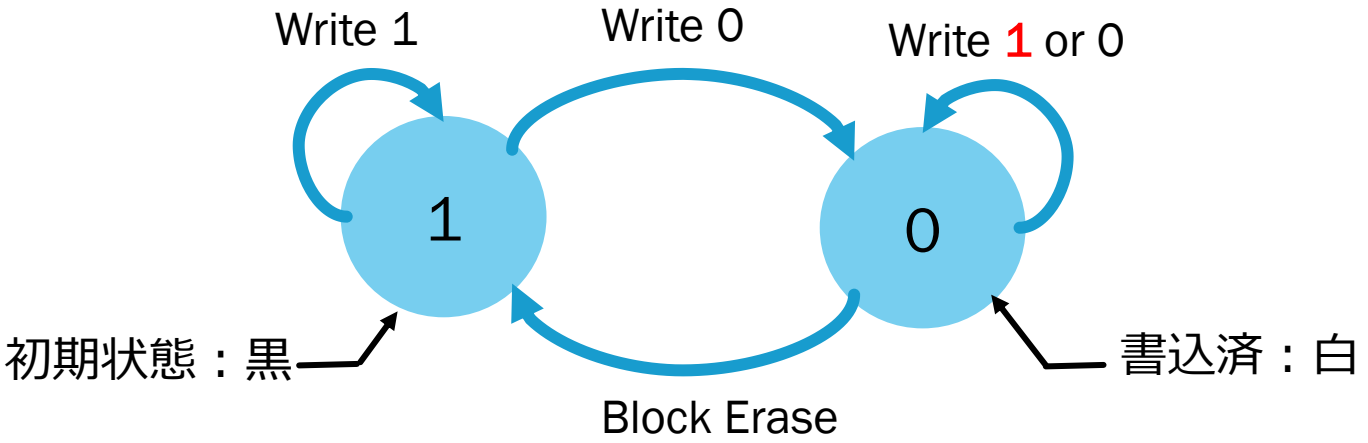
- フラッシュメモリ製コンピュータの紹介
- オセロコンピュータの開発
- オセロコンピュータを始める人へのアドバイス

フラッシュメモリの原理



- フラッシュメモリには、メモリセルという記憶素子が無数に集まっている
- （SLCの場合）初期状態が１。制御ゲートへ高電圧をかけ、電荷を溜めて0を書き込む
- 消去するときはシリコン基板に高電圧をかけて、電荷を抜く
- 電荷蓄積膜に電荷が溜まっているかどうかで閾値電圧が変わり、溜まっているほど電流が流れにくくなる（0と読める）

フラッシュメモリで計算するアイデア



- メモリセルは黒板のようなもの

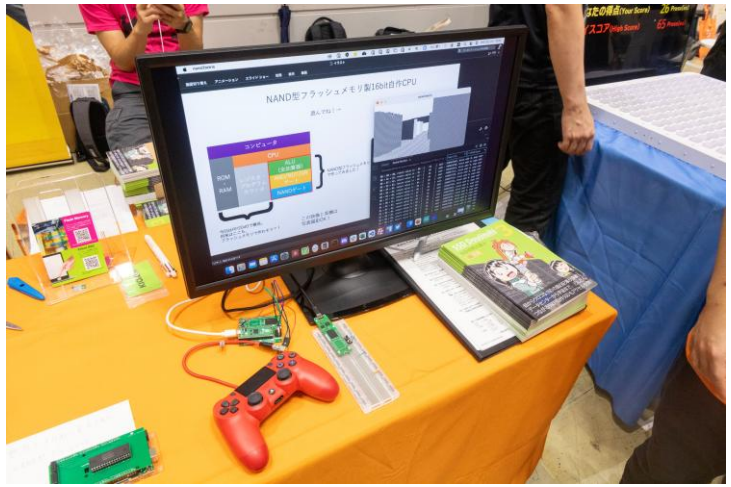
  →書き込みと消去の方法が違う、消去しなければ再利用不可
- あえてメモリセルに上書きすると、書いた通りのデータが読めない（当たり前）

  →読み出し値は1bit ANDと結果が同じ！（発見）

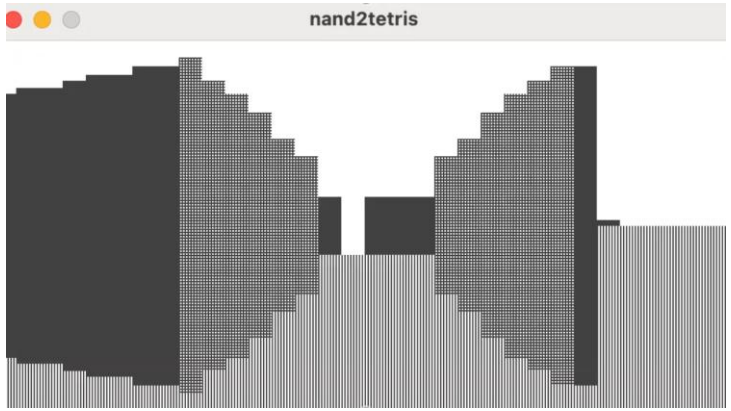
  →読み出し値をソフトで反転すればNAND演算ができそう

1回目の書込	2回目の書込	読出し値
0	0	0
0	1	0
1	0	0
1	1	1

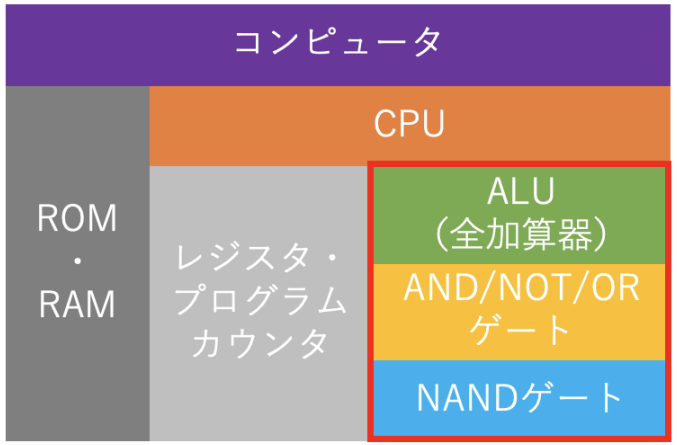
(2024年9月)フラッシュメモリ製CPUを作ってみた



https://x.com/yuuitirou528/status/1838130470959362271



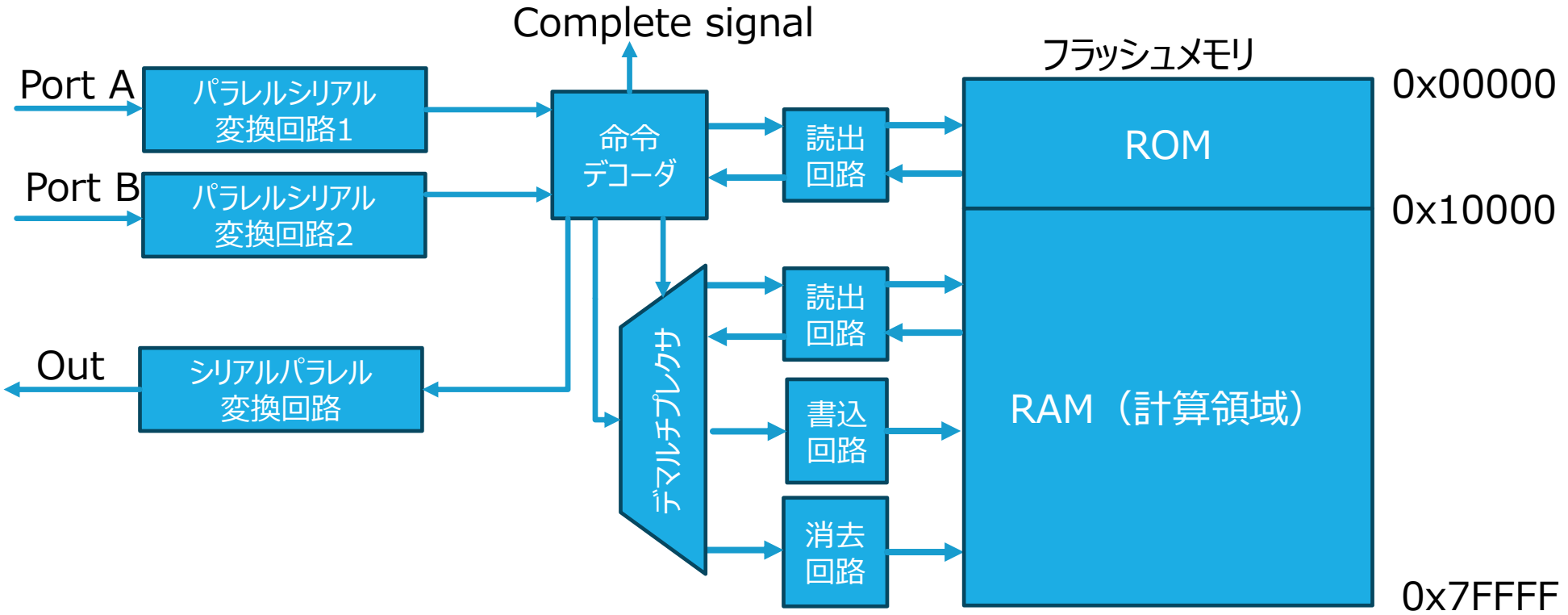
ゲームのプレイ画面(20倍速)



赤枠をフラッシュメモリで自作↑

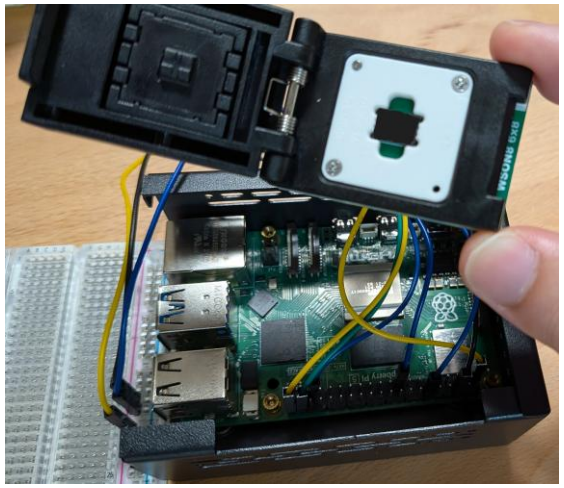
- nand2tetrisの16 bit加算回路をNAND型フラッシュメモリの計算で実現
- デモとして3D迷路ゲームを実行
- CPUの動作周波数はたった2Hz（！？）

(2024年12月) FMOP：Flash Memory Only Processorを考案

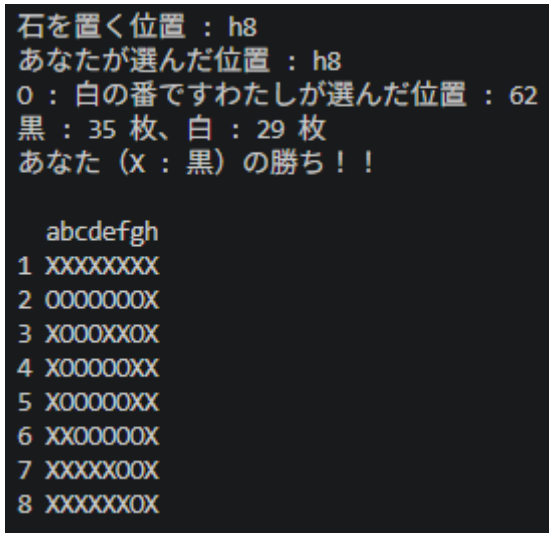


- フラッシュメモリとその読み書き消去回路のみで構成可能な、独自の命令セットを作ってみた
- フラッシュメモリの先頭領域をROM、後続領域をRAMとして使用。RAMは先頭から使い潰していく
- I/Oはパラシリ/シリパラ変換回路で1bitずつ入出力

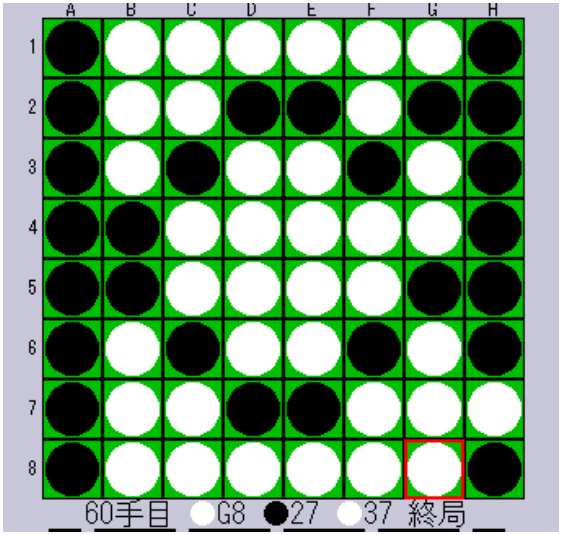
(2025年12月)フラッシュメモリ製オセロコンピュータ



実機環境（Raspberry Pi 5）



ゲームの対戦画面



対戦の様子[1]

- OISCのため高水準言語でプログラムを記述可能
- CPUの動作周波数は30.76 Hz まで向上！！（約16倍）
- まだまだ弱いが、まぐれで人間に勝てることもある

[1] https://do2.rojo.jp/shogi/javascript/mania.html

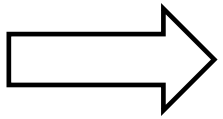
One Instruction Set Computer

```
*b = *b - *a; if(*b <= 0) goto c;
```

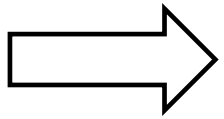
- 命令が1種類のコンピュータ。命令デコーダさえ不要！
- Subleq：OISCの1種で、なんと **チューリング完全**
- 減算器と負数判定は加算器のちょい変で作れる！
- コンパイラやアセンブリがOSSにいっぱいある

Higher Subleq

```c
int puts(char * a);

void main()
{
    puts("Hello, World!\n");
}
```



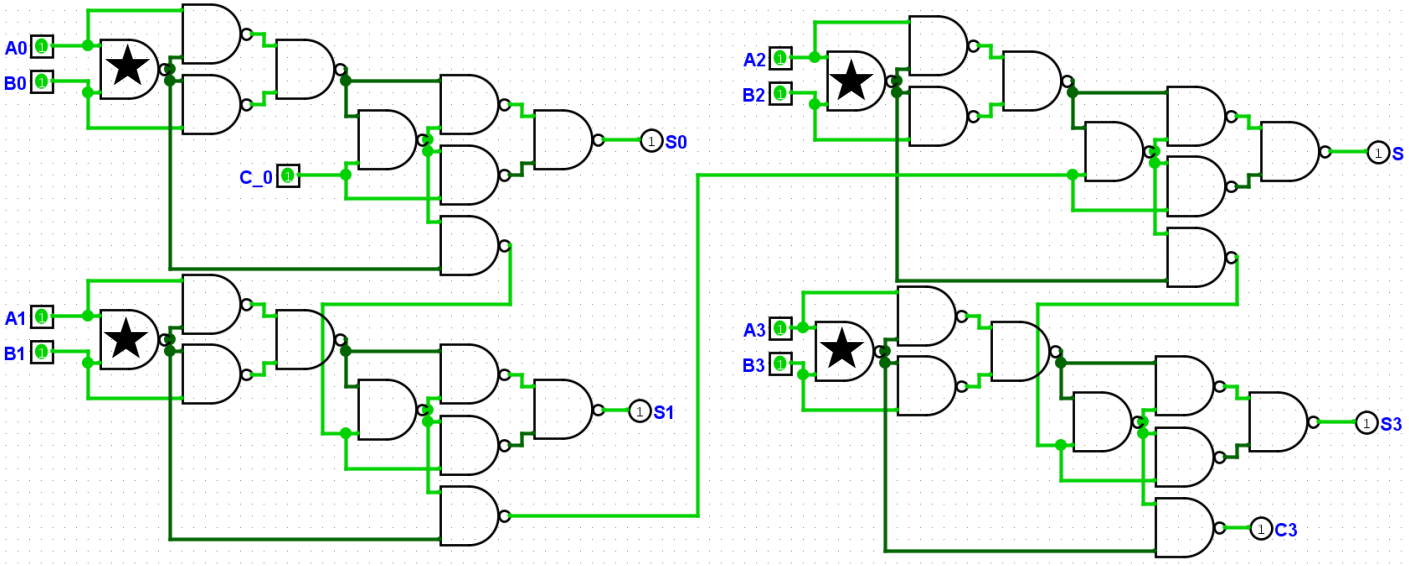
```
_main:
    dec sp; ?+11; sp ?+7; ?+6; sp ?+2; 0
    ?+6; sp ?+2; bp 0
    bp; sp bp
    dec sp; ?+11; sp ?+7; ?+6; sp ?+2; 0
    ?+6; sp ?+2; t1 0

    dec sp; ?+11; sp ?+7; ?+6; sp ?+2; 0
    ?+9; sp ?+5; c2 Z; Z 0; Z
    dec sp; ?+11; sp ?+7; ?+6; sp ?+2; 0
    ?+6; sp ?+2; ?+2 0 _puts; . ?;
    c3 sp

    ?+8; sp ?+4; t1; 0 t1; inc sp
    sp; bp sp
    ?+8; sp ?+4; bp; 0 bp; inc sp
    ?+8; sp ?+4; ?+7; 0 ?+3; Z Z 0

_puts:
    dec sp; ?+11; sp ?+7; ?+6; sp ?+2; 0
    ?+6; sp ?+2; bp 0
    bp; sp bp
    dec sp
    dec sp; ?+11; sp ?+7; ?+6; sp ?+2; 0
```

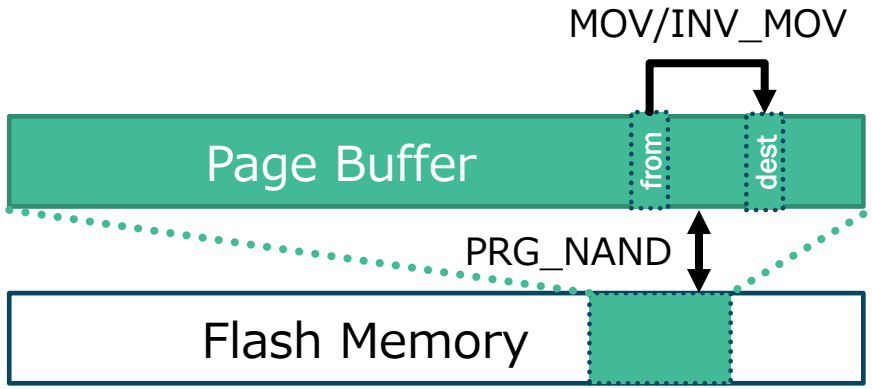


```
0 0 781
839 842 6
18 18 9
842 18 12
19 19 15
842 19 18
0 0 21
28 28 24
842 28 27
841 0 30
841 841 33
842 841 36
839 842 39
51 51 42
842 51 45
52 52 48
842 52 51
```

- C/C++言語風の構文
- 変数宣言、配列宣言、初期化、ポインタ宣言、文字列定数をサポート
- 関数呼び出しと最低限のライブラリ（標準入出力）をサポート

16倍高速化：バッファによるNAND演算の並列化

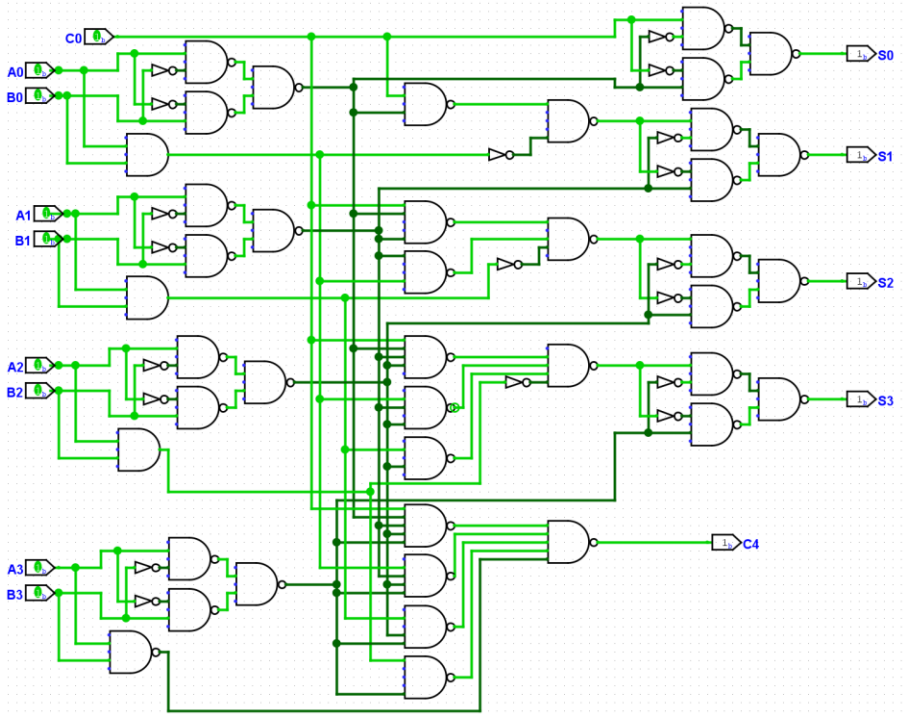




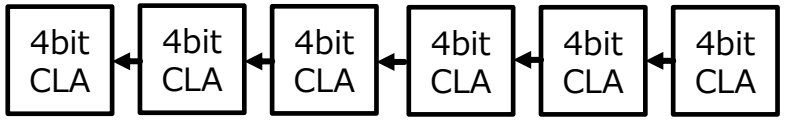
- 従来：ゲート1個につき2回Writeを実行
- 改善：Write 1回で2kB書き込むので、例えば★は同時に計算可能
- （余談）logisimはゲートレベルの設計にとても便利！

	4bit RCA	16bit RCA
従来	76回	304回
改善	14回	37回

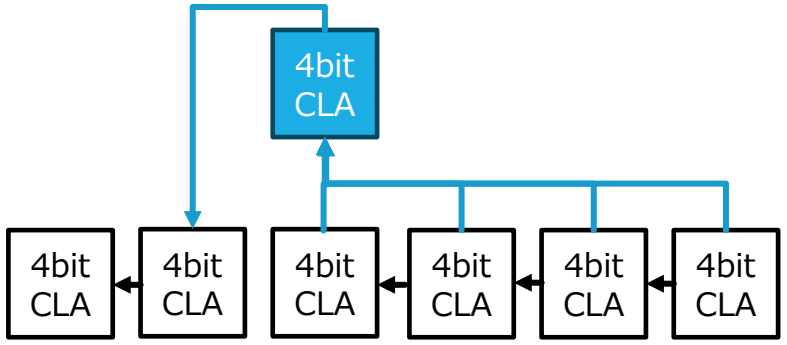
16倍高速化：階層型CLA(Carry Lookahead Adder)の実装



○CLA



○階層型CLA

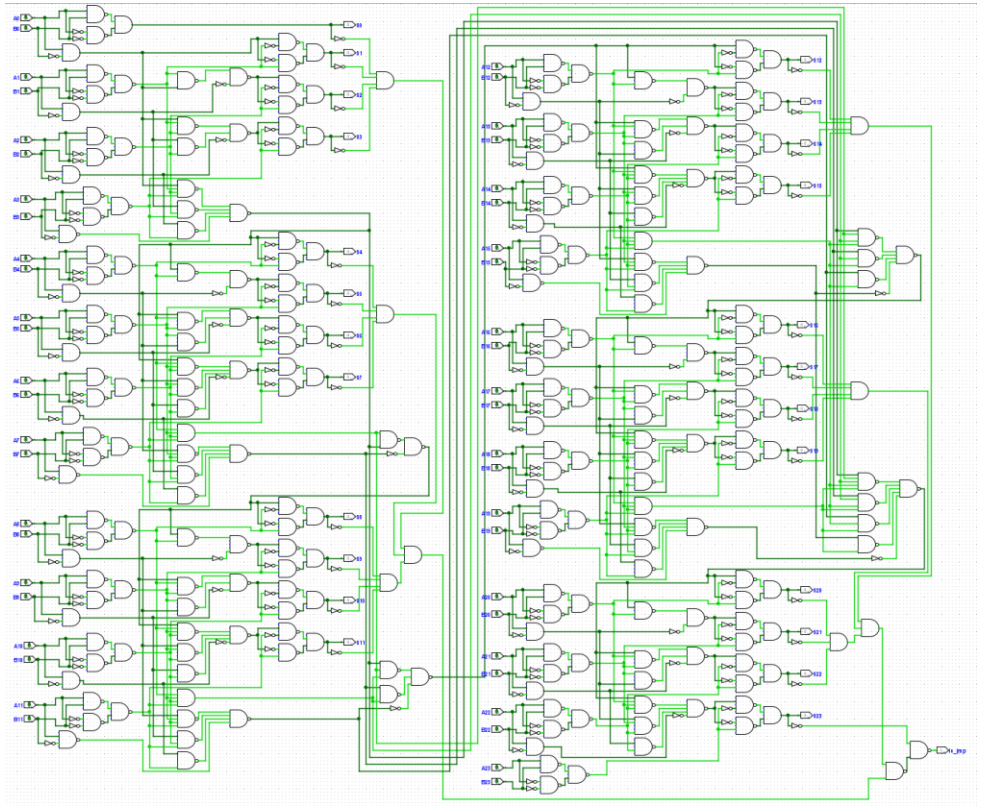


- RCAだとbit AND計算並列化の恩恵が少ない→4bit CLAで更に並列化
- 階層型CLAで更なる高速化を狙ったが、効果は少なかった

  ↑ファンイン数の増大がそのままクリティカルパスの増大に繋がるため

	4bit	16bit	24bit
RCA	14回	37回	53回
CLA	10回	16回	20回
HCLA	-	-	19回

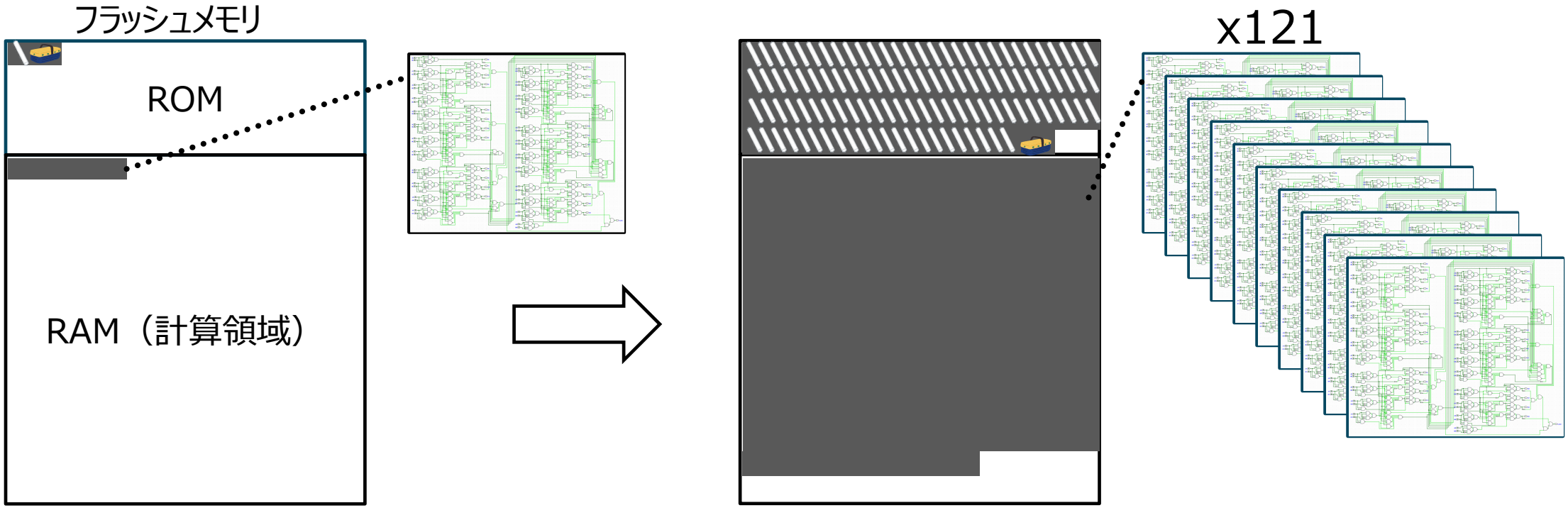
16倍高速化：階層型CLA(Carry Lookahead Adder)の実装



- 階層型CLAの解説は、前ページのように概念ブロック図で省略されがち（勉強に困った）
- 折角なのでANDとNANDだけで描いた24 bit CLAを共有※

※上図は加算器ではなく減算器ですが、ご愛嬌

16倍高速化：メモリ消去の抑制



- 従来：減算1回につき消去1回
- 改善：減算121回につき消去1回

→消去操作によるレイテンシと消耗を低減！→1210万 cycle（≒4.5日）以上稼働できそう！

オセロコンピュータの開発：盤面の管理

```
6   int raw_data[] = "\n  abcdefgh\n1 ........\n2 ........\n3 ........\n4 ........\n5 ........\n6 ........\n7 ........\n8 .....
7   int *board[64];
8
9   int BLACK = 88;
10  int WHITE = 79;
11
12  void init()
13  {
14    board[0]  = &raw_data[14];
15    board[1]  = &raw_data[15];
16    board[2]  = &raw_data[16];
17    ...
18    board[62] = &raw_data[97];
19    board[63] = &raw_data[98];
20  }
21
22  void setupGame()
23  {
24    *board[27] = WHITE;
25    *board[28] = BLACK;
26    *board[35] = BLACK;
27    *board[36] = WHITE;
28  }
29
30  void draw()
31  {
32    puts(raw_data);
33  }
```

	a	b	c	d	e	f	g	h
1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
3	16	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30	31
5	32	33	34	35	36	37	38	39
6	40	41	42	43	44	45	46	47
7	48	49	50	51	52	53	54	55
8	56	57	58	59	60	61	62	63

・ASCIIの1次元配列で盤面の管理と表示を兼ねる（黒はX:88、白はO:79）

・オセロの棋譜は、マスの座標をa1, b1, c1, … g8, h8と表現するのが一般的

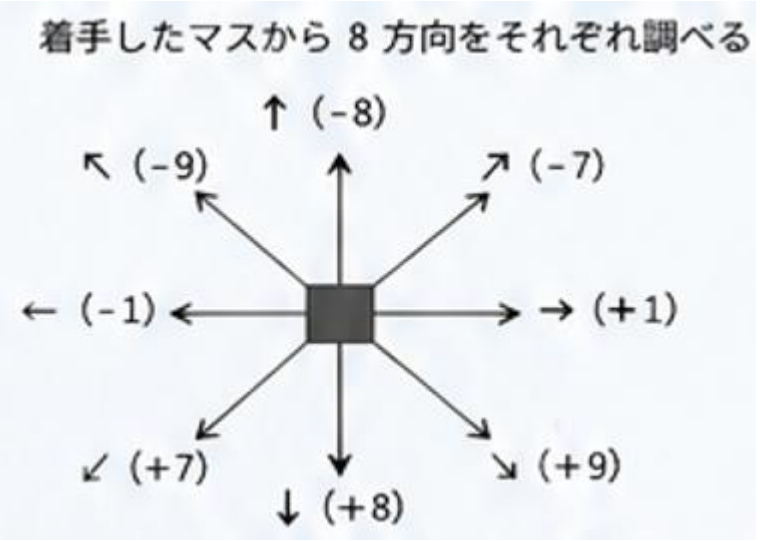
・ここでは a1=0, b1=1, c1=2, … g8=62, h8=63 とする

オセロコンピュータの開発：合法手の判定処理

```
353  int isValidPositionExist(int isBlack)
354  {
355    if(turn_num < 8)
356    {
357      turn_num++;
358      return 1;
359    }
360    for(int i = 0; i < 60; i++)
361    {
362      if(candidate[i] < 64)
363      {
364        if(evaluate(isBlack, candidate[i]))
365        {
366          turn_num++;
367          return 1;
368        }
369      }
370    }
371    return 0;
372  }
```

```
340  int evaluate(int isBlack, int num)
341  {
342    int reward = 0;
343
344    for(int i = 0; i < 8; i++)
345    {
346      rewards[i] = evaluate_direction(isBlack, num, i);
347      reward += rewards[i];
348    }
349
350    return reward;
351  }
```

	a	b	c	d	e	f	g	h
1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
3	16	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30	31
5	32	33	34	35	36	37	38	39
6	40	41	42	43	44	45	46	47
7	48	49	50	51	52	53	54	55
8	56	57	58	59	60	61	62	63



・手番は原則、黒→白の交互。ただし合法手がない（石が置けない）場合は相手に手番が移る

→毎ターン開始時、合法手の探索が必要（ただし8手目までは必ず合法手が存在するらしいのでSKIP可）

・合法手の探索は盤面のFor文。合法手かどうかの判定はさらに8方向のFor文

オセロコンピュータの開発：手番の実行

```
382  void execute(int isBlack, int num)
383  {
384      if(num < 64)
385      {
386          skip_count = 0;
387          for(int i = 0; i < 8; i++)
388          {
389              reverse(isBlack, num, direction[i], rewards[i]);
390          }
391      }
392  }
```

```
366  int reverse(int isBlack, int num, int direction_num, int reward)
367  {
368      for(int i = 0; i <= reward; i++)
369      {
370          if(isBlack)
371          {
372              *board[num] = 88;
373          }
374          else
375          {
376              *board[num] = 79;
377          }
378          num -= direction_num;
379      }
380  }
```

・合法手かどうかの判定のため、コマを何枚までひっくり返せるか数える（これを“報酬の計算”と呼ぶことにする）

・黒（＝プレイヤー）の手番：選択されたマスの報酬を計算→1以上なら手を確定

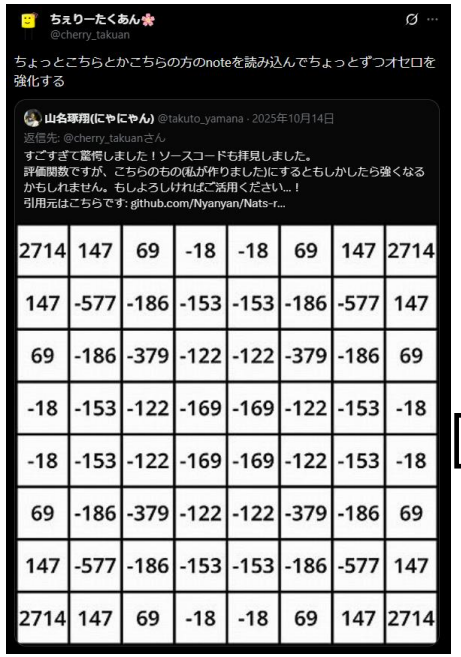
・白（＝CPU）の手番：報酬が1以上のマスが存在するか、盤面を端から探索

↑報酬の計算は計算コスト大。CPUの先読み処理は妥協し、最初に発見したマスをCPUの手とする

オセロコンピュータの開発：CPUの戦略

・CPUの先読み処理が無くても勝ちたい→合法手の探索ルートを配列で定義、書き換え可能に設計

・「角マスを優先的に取る」、「相手が角マスを取れそうな手を避ける」など初歩的な定石は反映できた



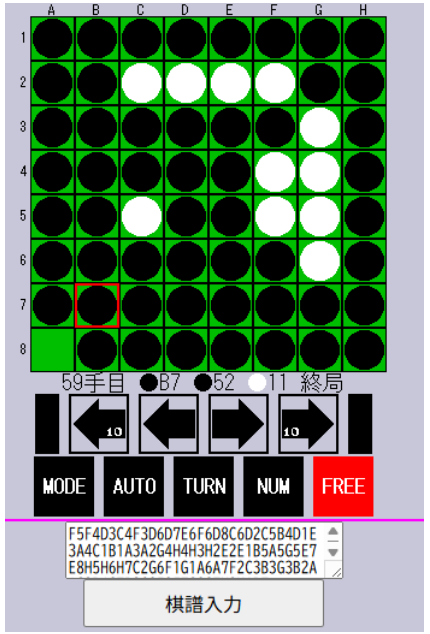
有力な情報[2]

	a	b	c	d	e	f	g	h
1	0	4	12	20	21	13	5	1
2	6	56	44	36	37	45	57	7
3	14	46	52	28	29	53	47	15
4	22	38	30	■	□	31	39	23
5	24	40	32	□	■	33	41	25
6	16	48	54	34	35	55	49	17
7	8	58	50	42	43	51	59	9
8	2	10	18	26	27	19	11	3

有力な情報[2]を探索ルート化

```
105    candidate[0] = 0;
106    candidate[1] = 7;
107    candidate[2] = 56;
108    candidate[3] = 63;
109    candidate[4] = 1;
110    candidate[5] = 6;
111    candidate[6] = 8;
112    candidate[7] = 15;
113    candidate[8] = 48;
114    candidate[9] = 55;
115    candidate[10] = 57;
```

探索ルートをコード化

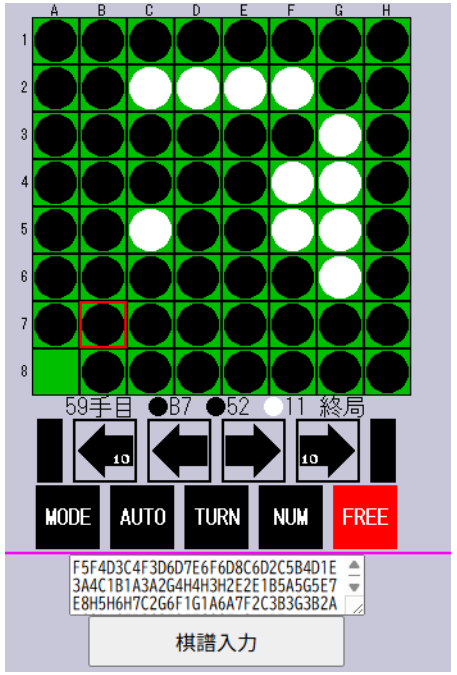


見事に惨敗（まだ弱い）

[2] https://github.com/Nyanyan/Nats-reversi

これからオセロコンピュータを始める人へ

- ゲームプログラムとして成立させることが第一！注意点は
  - 合法手が無い手番のスキップ
  - 互いに合法手が無い時の対局終了
- 動作テストをするならKaggleのデータセット[3]を読み込ませるのがお勧め
- 棋譜を可視化・再生するなら「オセ★まにあ」さん[4]がお勧め
  - Kaggleのデータセットもコピペ可能
  - 合法手をハイライトしてくれるので、CPUとの対局時もラクができる



[3] https://www.kaggle.com/datasets/andrefpoliveira/othello-games

[4] https://do2.rojo.jp/shogi/javascript/mania.html

まとめ

- フラッシュメモリ製コンピュータの概要を紹介
- オセロコンピュータの実装を紹介
- 今後の展望
  - フラッシュメモリ製CPUの高速化：時間制限順守のため
  - 先読みの実装：人間程度に強くするなら必要そう