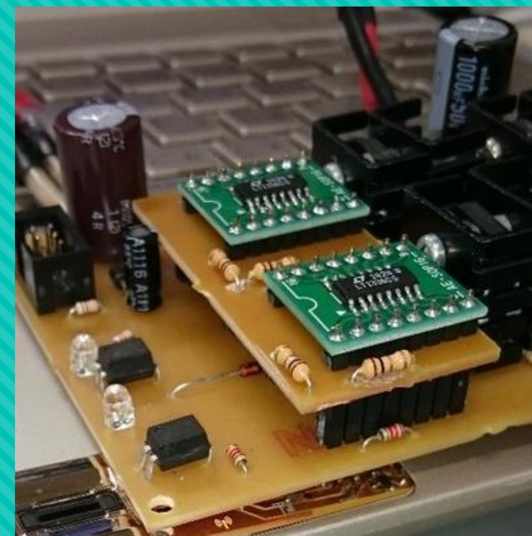
フラッシュメモリでCPUを創りたい

2024年12月01日 第4回自作CPUを語る会
13:15～13:45（初心者枠）
講演者：にちか(@lxacas)



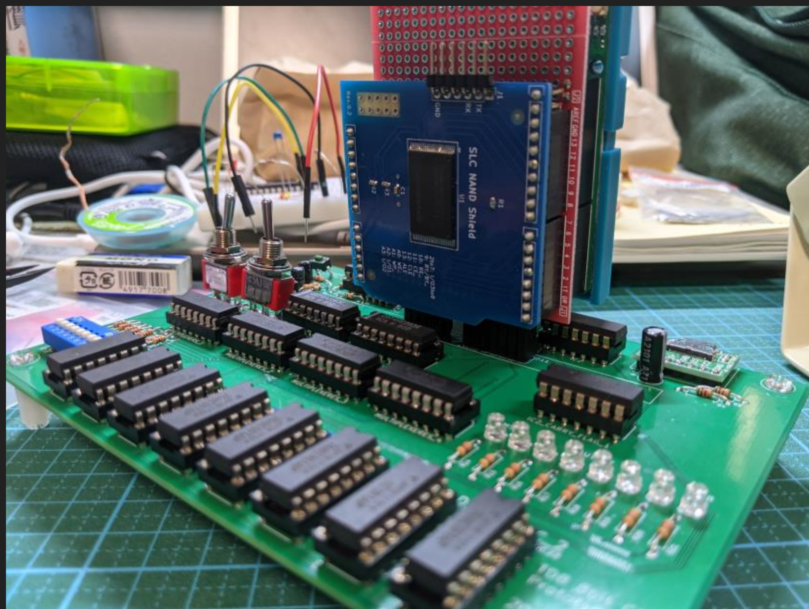
～NAND型フラッシュメモリがあればNANDEもできる！？～

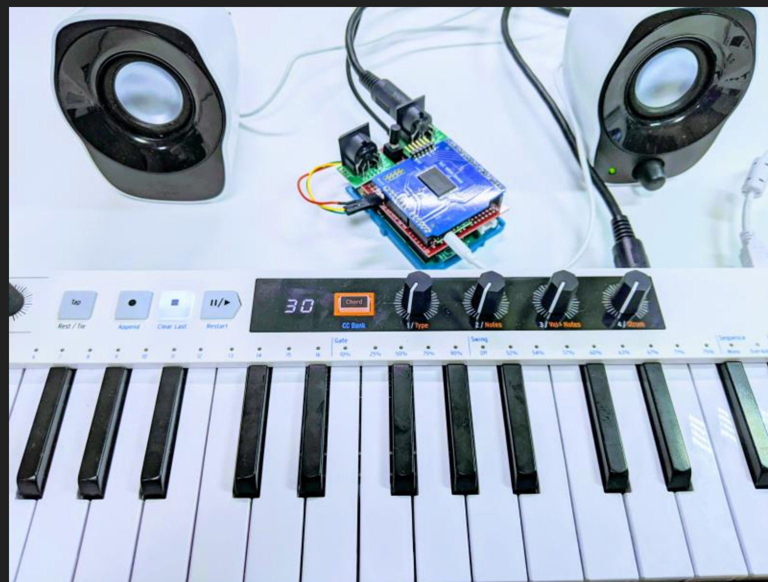
はじめに

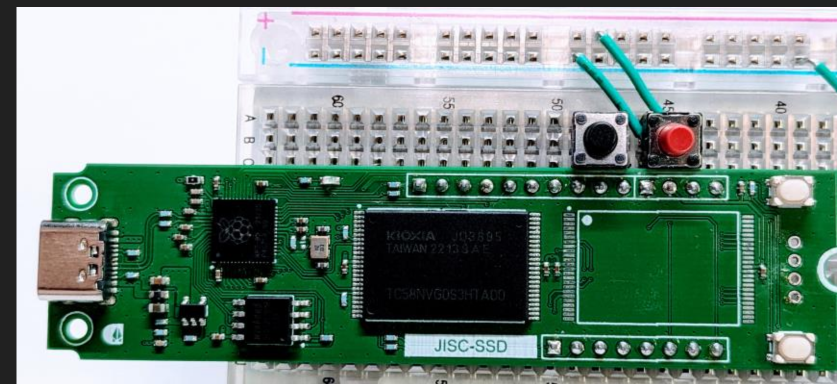
- キオクシアの人達と一緒に、SSD同人誌という冊子を書いてます！
- 第1号～2号は、以下のURLからDLできます！

自己紹介

- 趣味：電子工作、DIY
  - Maker Faire Tokyo 2021：自作CPUを自作SSDで動かすデモ（究極の疑問の答え：42を計算する）
  - Maker Faire Tokyo 2022：自作SSDを使ったMIDI録音デバイス





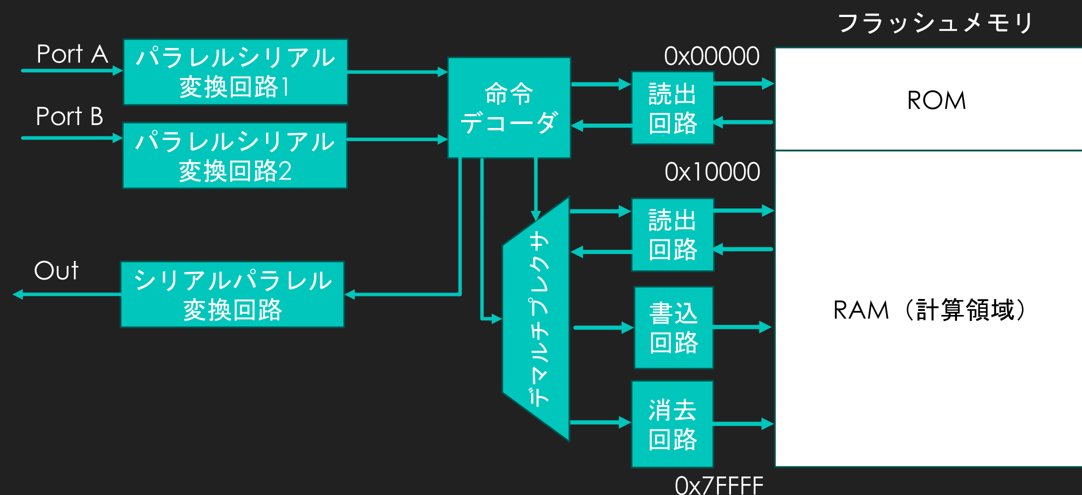


今日までに創ったもの

- Maker Faire Tokyo 2024（9/21~22）
  - nand2tetrisエミュレーター
  - 約2Hzで動作するNAND型フラッシュメモリ製ALU
- 第４回自作CPUを語る会（12/1本日）
  - フラッシュメモリの読み書きだけで構成するコンピュータのアーキテクチャ

  →名付けてFLash memory Only Processor System : FLOPS

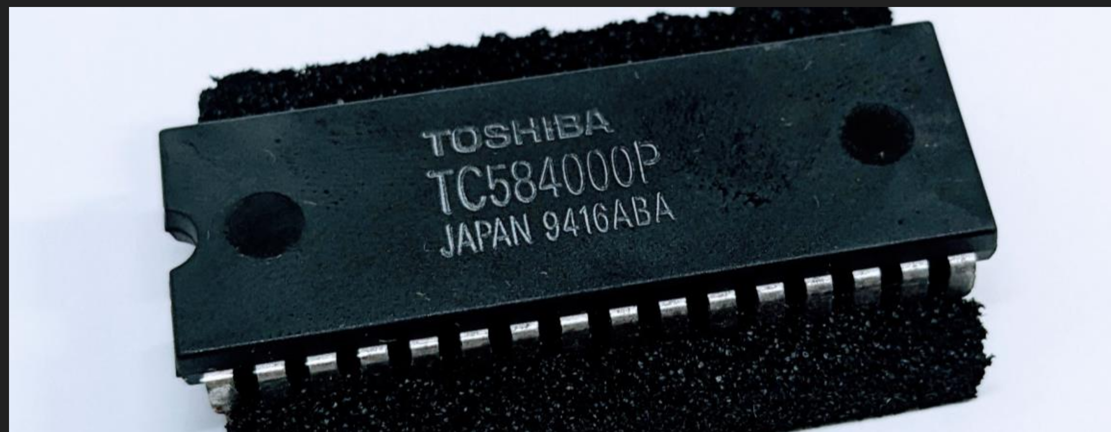
https://www.oreilly.co.jp/books/9784873117126/



なぜフラッシュメモリ？

- 鉄板のTD4とnand2tetrisを履修し、次はオリジナルな設計をしたい
- サークルの先輩に、「にちかくん、これあげるよ」と

「世界初のNAND型フラッシュメモリ」 を貰う

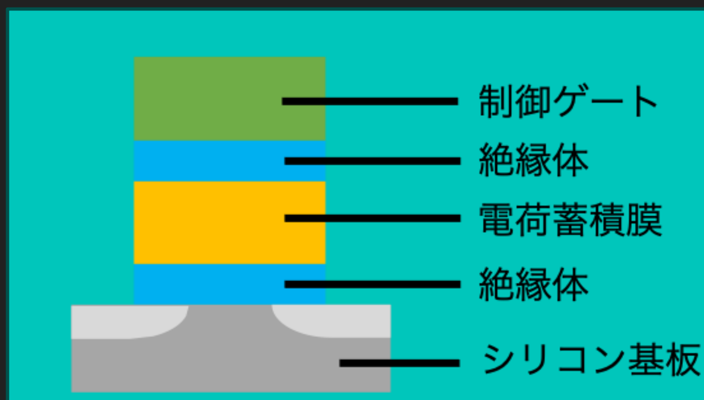


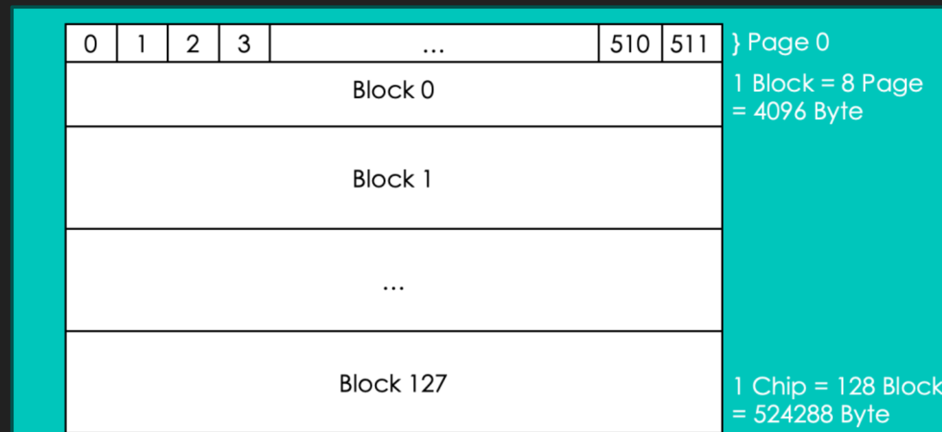
- これで何か面白いことができないだろうか？

フラッシュメモリの仕組み

※以降、諸元はTC584000Pをもとに説明

- 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。
- 初期状態が１。0を書き込む際に制御ゲートへ高電圧をかけ、電荷を溜める。
- 電荷蓄積膜に電荷が溜まっているかどうかで、電流が流れるか、流れないかが決まり、1 or 0と判定
- 消去する際、ゲートに書込時と逆方向に高電圧をかけて、電荷を抜く。

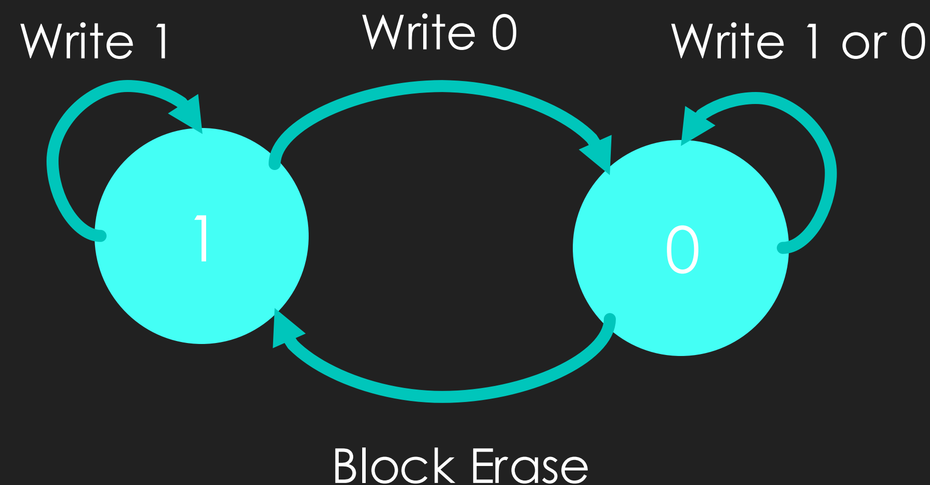




フラッシュメモリで計算するアイデア

フラッシュメモリ触っているうちに、以下の真理値表を閃く！
「これANDじゃない？」

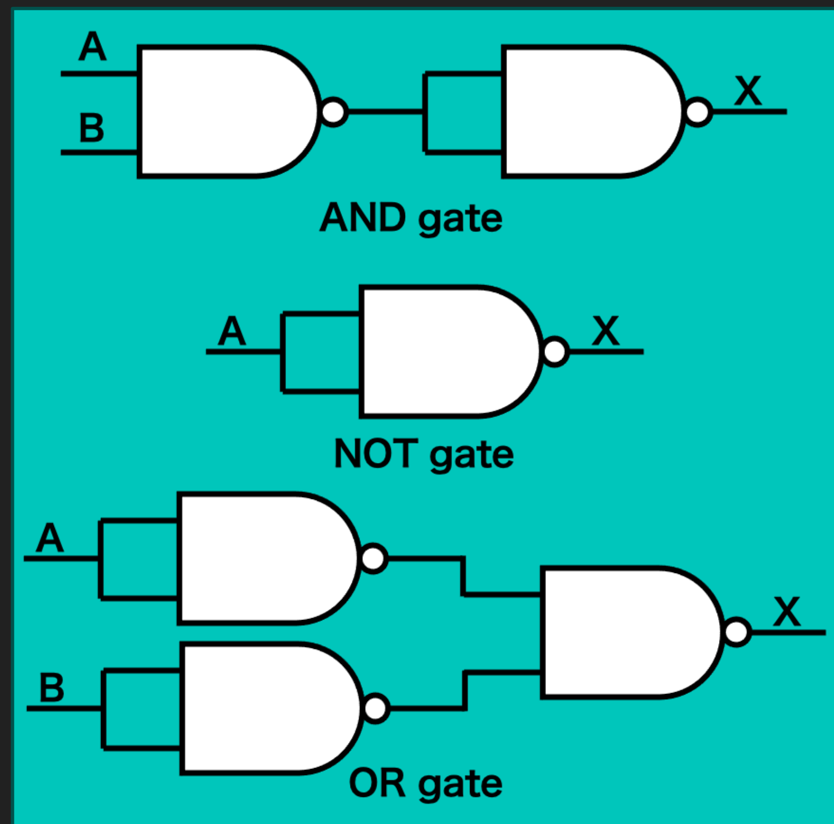
1回目の書込値	2回目の書込値	読出し値
0	0	0
0	1	0
1	0	0
1	1	1



フラッシュメモリで計算するアイデア

1回目の書込値	2回目の書込値	読出し値	読出し値の反転
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

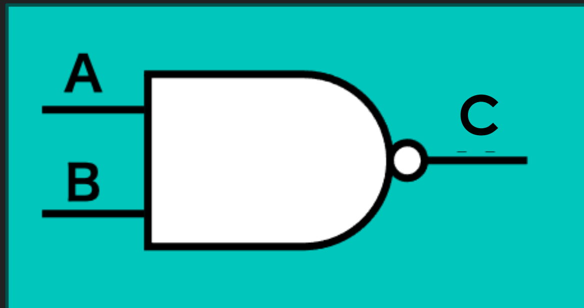
結果をひっくり返せば→NANDになる！



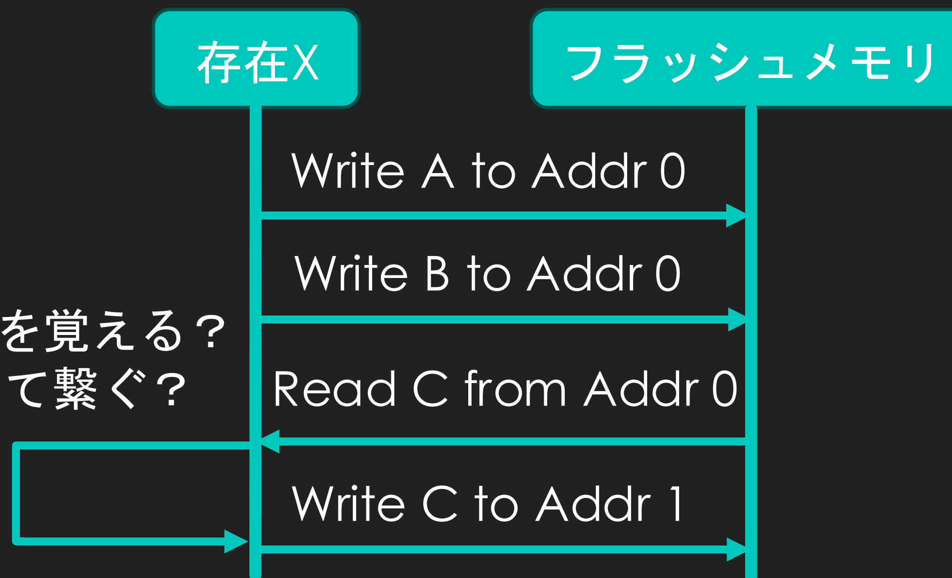
NANDがあれば→NANDEもできる！

NAND演算ができることは判ったけど…

- あるゲートの出力を、別のゲートの入力にしなければならない。
- フラッシュメモリは受動部品。データを書込み、読出し、次の書込みを行う存在Xが必要。
- フラッシュメモリをCPU（の部品）にしたコンピュータなんて見たことない。
- アーキテクチャーなんて設計したことがない、けれどもMFTの開催日はやってくる

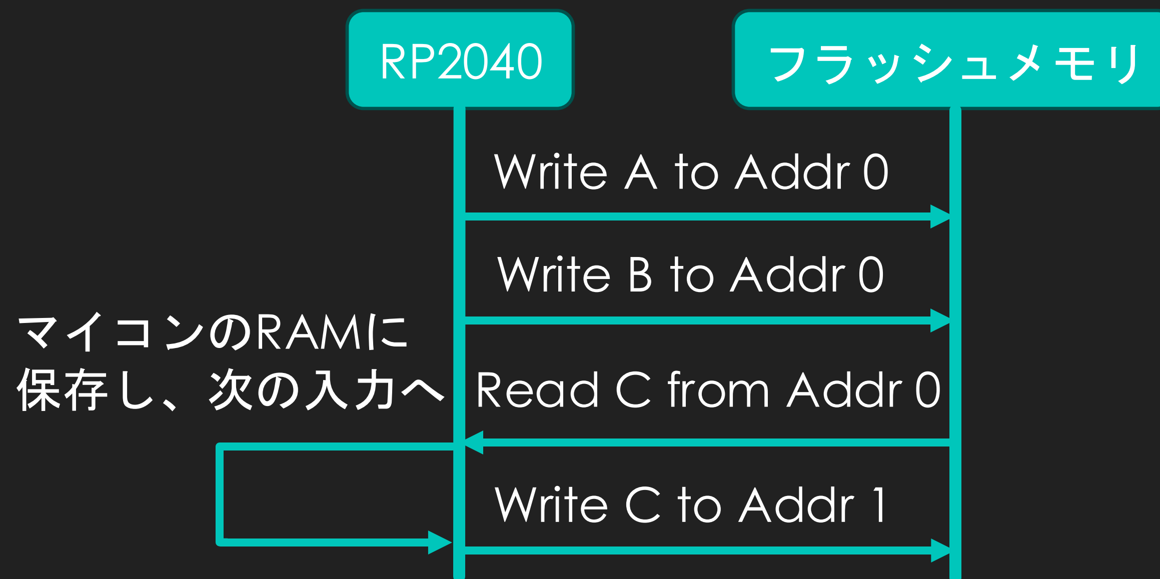


誰が値Cを覚える？
どうやって繋ぐ？



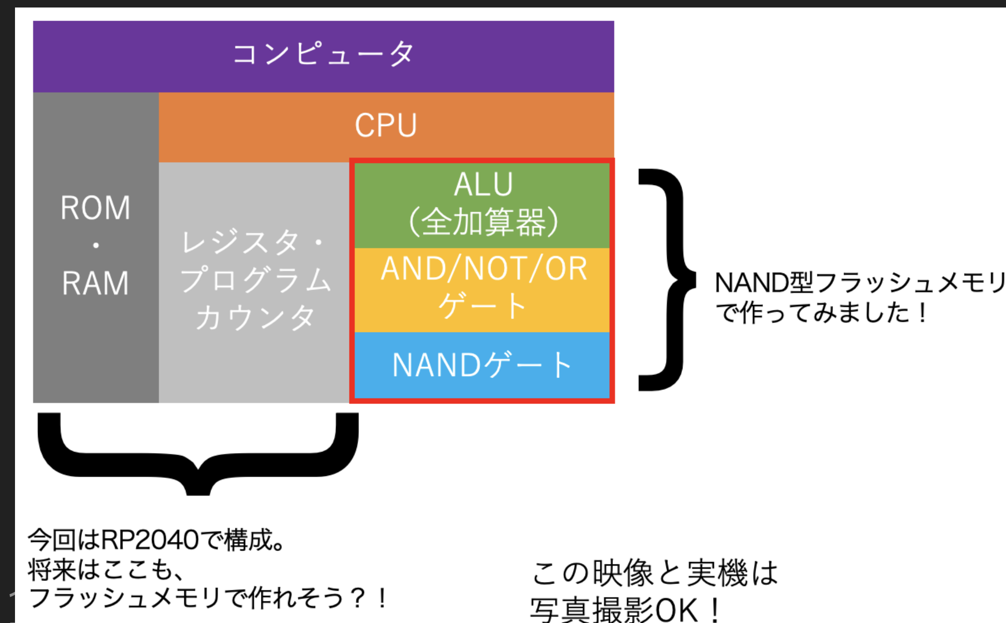
Maker Faire Tokyo 2024の方針

- まずは自作CPU内の加算・AND演算だけフラッシュメモリの読み書きで置き換えることを目指す
- 妥協策として、存在XにRaspberry Pi Picoを使う。
- アーキテクチャも、nand2tetrisを参考にする



Maker Faire Tokyo 2024の展示作

- Raspberry Pi Picoで、nand2tetrisのエミュレータを作成。
- ALUの処理だけ、NAND型フラッシュメモリに置き換え、約2Hzで動作を確認した。
- 理屈上は、リレー・真空管・トランジスタ、プラレール、折り紙など好きな部品でnand2tetrisができそう！



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

展示会は成功したけれども

- 結局高性能な既製CPUで、より性能の悪いCPUを組み立てているにすぎない

  →理想は、フラッシュメモリとCPU未満な小規模回路で、CPUを自作したい。

  →アーキテクチャの設計に挑戦してみよう！

- フラッシュメモリはフリップフロップのお化け。速度に目を瞑れば、レジスタやRAMだって作れるハズ。

  →All フラッシュメモリ製自作CPU…ってコト！？

設計方針

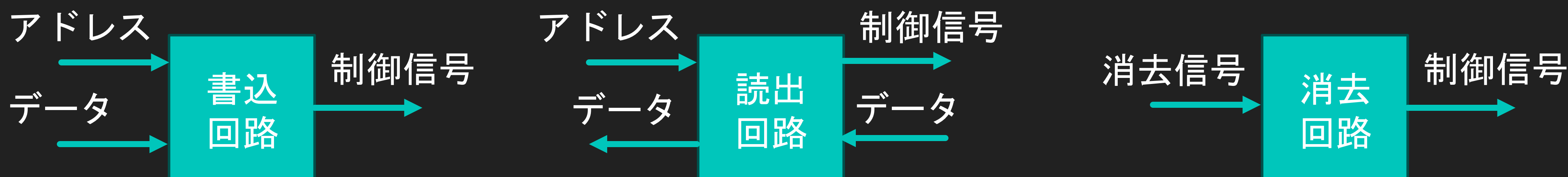
- フラッシュメモリは、一度0を書き込んでしまうと、消去しない限り戻せない
  - 多数のメモリセルを回路的に繋げる、並行処理は大変難しい。
  - MFTのときのように、愚直に1bitずつの読み書きを繰り返すしかない。

- 1bit のNAND演算回路でコンピュータは作れるのだろうか？
  - 幸い、ROMやRAMは潤沢に用意できる

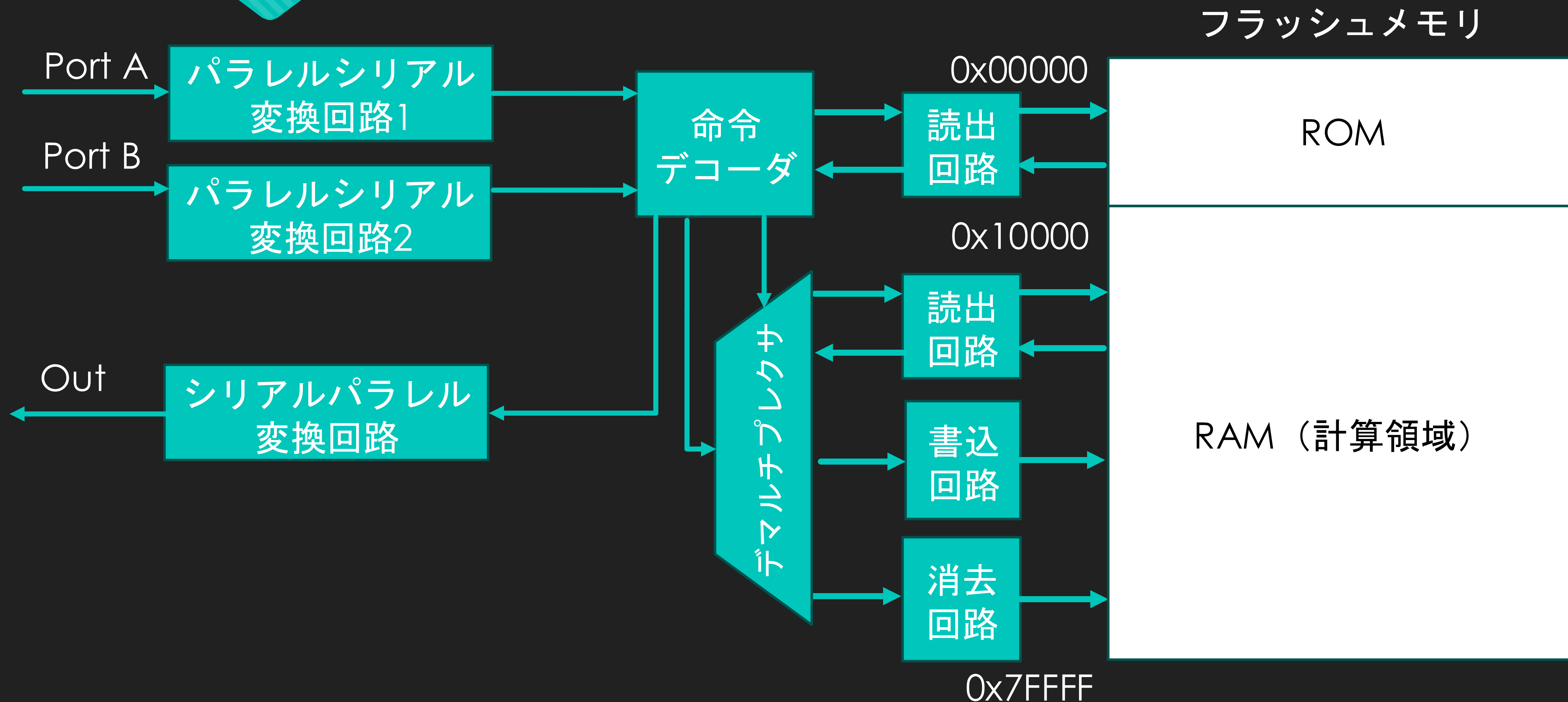
    →Let's プログラム内臓方式♪

魔法の回路に頼ろう

- 論理回路設計は初心者のため、現段階では回路の詳細設計を割愛する。
- フラッシュメモリの操作は、書込み・読出し・消去の3つ。これらができる論理回路が最初からあるものとする
- 上記の部品と都合の良い命令デコーダを使って、CPUのアーキテクチャを考えてみる。



FLash memory Only Processor System



命令セット

offset	63~61(3 bit)	60~57(4bit)	56~38(19 bit)	37~19(19 bit)	18~0(19 bit)
用途	命令コード	reserved	dest	from	next

←プログラムカウンタを
持ちたくないので
次のアドレスを埋める

63 bit	62 bit	61 bit	命令	説明
0	0	0	MOV	fromからデータを読み、destへ書き込む。All 0 の場合は実質NOP。
0	0	1	INV_MOVE	fromからデータを読み、反転してからdestへ書き込む
0	1	0	CAP_IN	入力ポートA,Bの値をシフトレジスタに取り込む
0	1	1	GET_IN	入力ポートAのLSBをfromに、BのLSBをdestに書き込む
1	0	0	PUSH_OUT	fromのbitで多数決を取り、過半数が0なら0を、1なら1を出力ポートのシフトレジスタへ転送する
1	0	1	JMP	T.B.D.
1	1	0	RAISE	FLOPSの周辺回路を1クロック進める（計算完了タイミングを外部に知らせる）
1	1	1	ERASE	RAM領域を消去して初期化する

メモリマップ

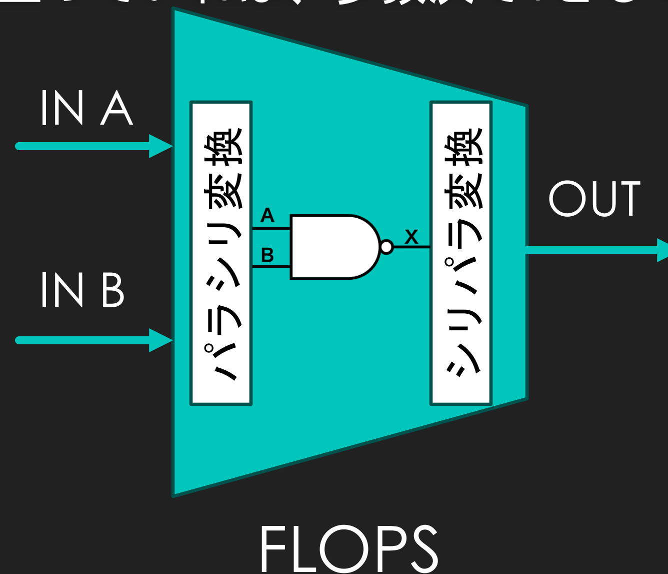
- ROM：命令を格納。1WORD = 8 Byte。8192命令まで格納可能。
- RAM：NAND演算を行う場であり、かつ途中結果を保存しておく場所。
- RAMとして確保された領域を使い切ったら、ブロック消去が必要。
- プログラムカウンタも含めてレジスタは存在しない。

アドレス	領域
0x00000	ROM
0x10000	RAM（計算領域）
0x7FFFF	

I/Oポート

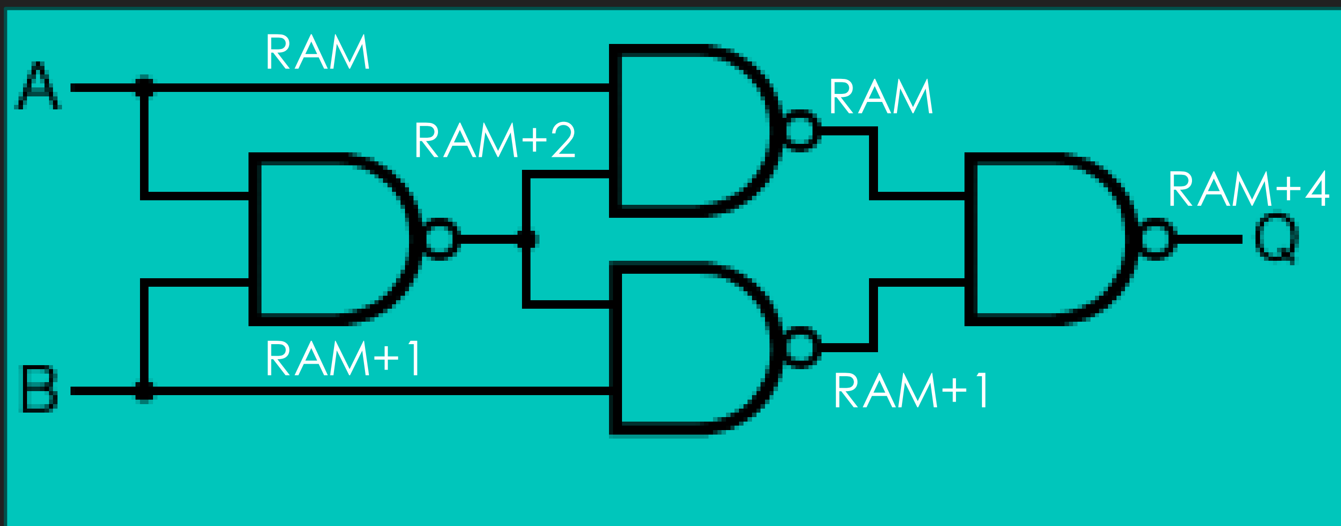
- 入力ポートはA, Bの2つ、出力ポートは1つ（いずれも16 bit幅）
- 入力ポート：一度に1bitのNANDしか計算ができないので、パラレル-シリアル変換回路を挟む
- 出力ポート：一度に1bitずつしか計算が終わらないので、シリアル-パラレル変換回路を挟む
- 実際のフラッシュメモリは一定確率で誤るので、0x01は0xffとして読み書きする

→読み出し時に過半数のbitが立っていれば、多数決で1として扱う。例）0b11101010→trueと判定



ROM

- 1bit加算の実行例を示す。

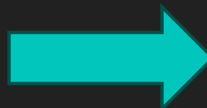



```
69  ROM_t rom[ROM_NUM] = {
70      //Erase RAM area
71      {ERASE, 0, 0, 0, 0x0008},
72      //Capture inputs
73      {CAP_IN, 0, 0, 0, 0x0010},
74      {GET_IN, 0, RAM, RAM+1, 0x0018},
75      //Add LSB
76      {MOV, 0, RAM+2, RAM, 0x0020},
77      {MOV, 0, RAM+2, RAM+1, 0x0028},
78      {INV_MOV, 0, RAM, RAM+2, 0x0030},
79      {INV_MOV, 0, RAM+1, RAM+2, 0x0038},
80      {INV_MOV, 0, RAM+3, RAM, 0x0040},
81      {INV_MOV, 0, RAM+3, RAM+1, 0x0048},
82      {INV_MOV, 0, RAM+4, RAM+3, 0x0050},
83      {PUSH_OUT, 0, 0, RAM+4, 0x0058},
84      //raise signal
85      {RAISE, 0 , 0, 0, 0x0060}
86      };
```

エミュレータを書いてみた

- C++言語で書いてみた。（一昨日）
- ROMは、vecotr型変数と関数を利用して簡易的に生成できるようにした（今朝５時）

```cpp
69  ROM_t rom[ROM_NUM] = {
70      //Erase RAM area
71      {ERASE, 0, 0, 0, 0x0008},
72      //Capture inputs
73      {CAP_IN, 0, 0, 0, 0x0010},
74      {GET_IN, 0, RAM, RAM+1, 0x0018},
75      //Add LSB
76      {MOV, 0, RAM+2, RAM, 0x0020},
77      {MOV, 0, RAM+2, RAM+1, 0x0028},
78      {INV_MOV, 0, RAM, RAM+2, 0x0030},
79      {INV_MOV, 0, RAM+1, RAM+2, 0x0038},
80      {INV_MOV, 0, RAM+3, RAM, 0x0040},
81      {INV_MOV, 0, RAM+3, RAM+1, 0x0048},
82      {INV_MOV, 0, RAM+4, RAM+3, 0x0050},
83      {PUSH_OUT, 0, 0, RAM+4, 0x0058},
84      //raise signal
85      {RAISE, 0 , 0, 0, 0x0060}
86  };
```



```cpp
49  void rom_init(std::vector<ROM_t>& rom)
50  {
51      //Erase RAM area
52      rom.push_back({ERASE, 0, 0, 0, incAddr()});
53      //Capture inputs
54      rom.push_back({CAP_IN, 0, 0, 0, incAddr()});
55      uint32_t carry_in = getCalcAddr();
56      //Flip the 1st carry bit to 0.
57      rom.push_back({INV_MOV, 0, carry_in, carry_in, incAddr()});
58      for(int i = 0; i < DIGIT; i++)
59      {
60          uint32_t carry_out = getCalcAddr();
61          fullAdd(rom, carry_out, carry_in);
62          carry_in = carry_out;
63      }
64      //raise signal
65      rom.push_back({RAISE, 0 , 0, 0, 0x00});
66  }
```

動いた！

```
 4  int main(void)
 5  {
 6      CPU cpu;
 7      for(int i = 0; i < 127; i++)
 8      {
 9          for(int j = 0; j < 127; j++)
10          {
11              cpu.inA = i;
12              cpu.inB = j;
13              for(size_t k = 0; k < cpu.getRomSize(); k++)
14              {
15                  cpu.execute(cpu.fetch());
16                  if(cpu.signal)
17                  {
18                      if((i + j) == cpu.out)
19                      {
20                          printf("[  OK  ] %d = %d + %d\n", cpu.out, i ,j);
21                      }
22                      else
23                      {
24                          printf("[FAILED] %d = %d + %d\n", cpu.out, i ,j);
25                      }
26                  }
27              }
28          }
29      }
30      return 0;
31  }
```

デバッグ コンソール (⇧⌘Y)

ターミナル　ポート　問題 11　出力　デバッグ コンソール　シリアル モニター　MEMORY　XRT

```
[  OK  ] 216 = 121 + 95
[  OK  ] 217 = 121 + 96
[  OK  ] 218 = 121 + 97
[  OK  ] 219 = 121 + 98
[  OK  ] 220 = 121 + 99
[  OK  ] 221 = 121 + 100
[  OK  ] 222 = 121 + 101
[  OK  ] 223 = 121 + 102
[  OK  ] 224 = 121 + 103
[  OK  ] 225 = 121 + 104
[  OK  ] 226 = 121 + 105
[  OK  ] 227 = 121 + 106
[  OK  ] 228 = 121 + 107
[  OK  ] 229 = 121 + 108
[  OK  ] 230 = 121 + 109
[  OK  ] 231 = 121 + 110
[  OK  ] 232 = 121 + 111
[  OK  ] 233 = 121 + 112
```

- 16bit加算が正しく行えることを確認できた！
- 超大雑把にRead/Write/Erase時の処理時間を考慮すると加算1回に0.0966秒。約10.4Hz。最適化は未実施。

```
25  void CPU::erase(void)
26  {
27      DEBUG_PRINT("erase nand flash...\n");
28      memset(&nand[RAM], 0xff, sizeof(nand) - ROM_SIZE);
29      romAddr = 0;
30      calcAddr = RAM;
31      usleep(10000);
32  }
33
34  uint16_t CPU::read(uint32_t addr)
35  {
36      //DEBUG_PRINT("%02x = read(%07x)\n",nand[addr], addr);
37      usleep(15);
38      return nand[addr];
39  }
40
41  void CPU::write(uint32_t addr, uint8_t value)
42  {
43      DEBUG_PRINT("write %07x, %02x\n", addr, value);
44      nand[addr] &= value;
45      usleep(40);
46  }
```

まとめ

- フラッシュメモリでNANDを計算する方法を思いついた
- NAND型フラッシュメモリとRaspberry Pi Picoを用いて、2Hzで計算できることを実機確認した
- 1bit NANDしか計算できないALUで、コンピュータの構成方法を検討した
  - アーキテクチャ、命令セットを設計した
- エミュレータを開発し、16bitの足し算が動作することを確認できた

さいごに

- キオクシアの人達と一緒に、SSD同人誌という冊子を書いてます！
- 第1号～2号は、以下のURLからDLできます！
- **第３号は40冊持ってきました！ぜひお手にとってください！！**