NANDによるプロセッサ開発

ちぇりー技術 ちぇりーたくあん

Twitter : @cherry_takuan

はじめに ～NLP-16の紹介～

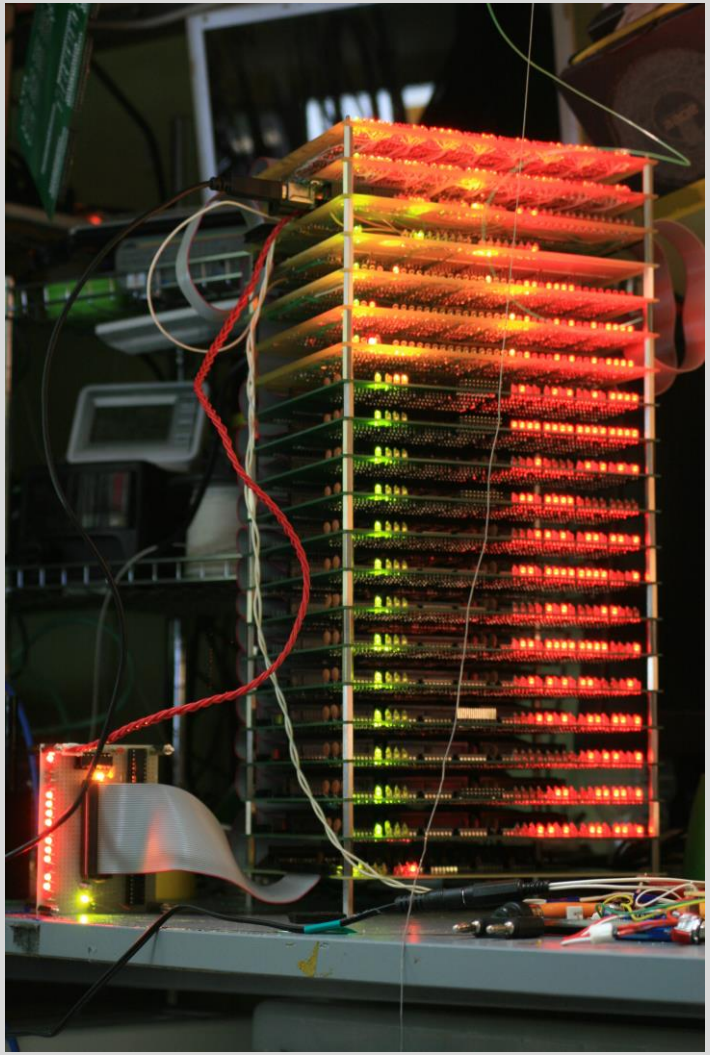
正式名称　「NAND Logic Processor – 16bit processor」

2入力NAND素子のみで作られた16bitのCPU

命令はインテルの初期のMPU, i8080に近い命令を実行可能

目標は黎明期のMPUに匹敵する処理機能を持つプロセッサをNAND素子のみで独自に開発すること

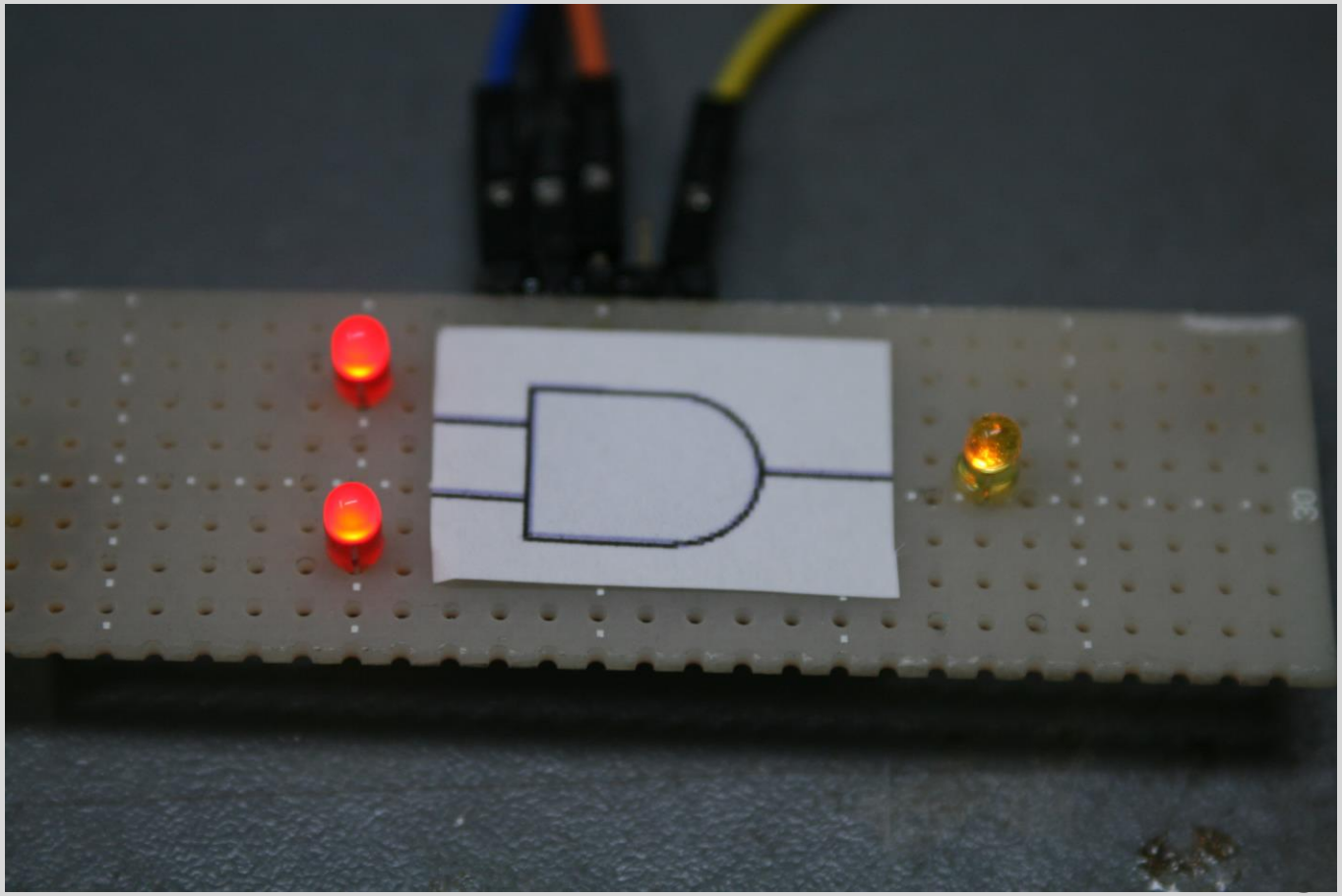
現在は次世代機「NLP-16A」を開発中



NANDとはNANDぁ(何だ)？

基本の論理「AND」

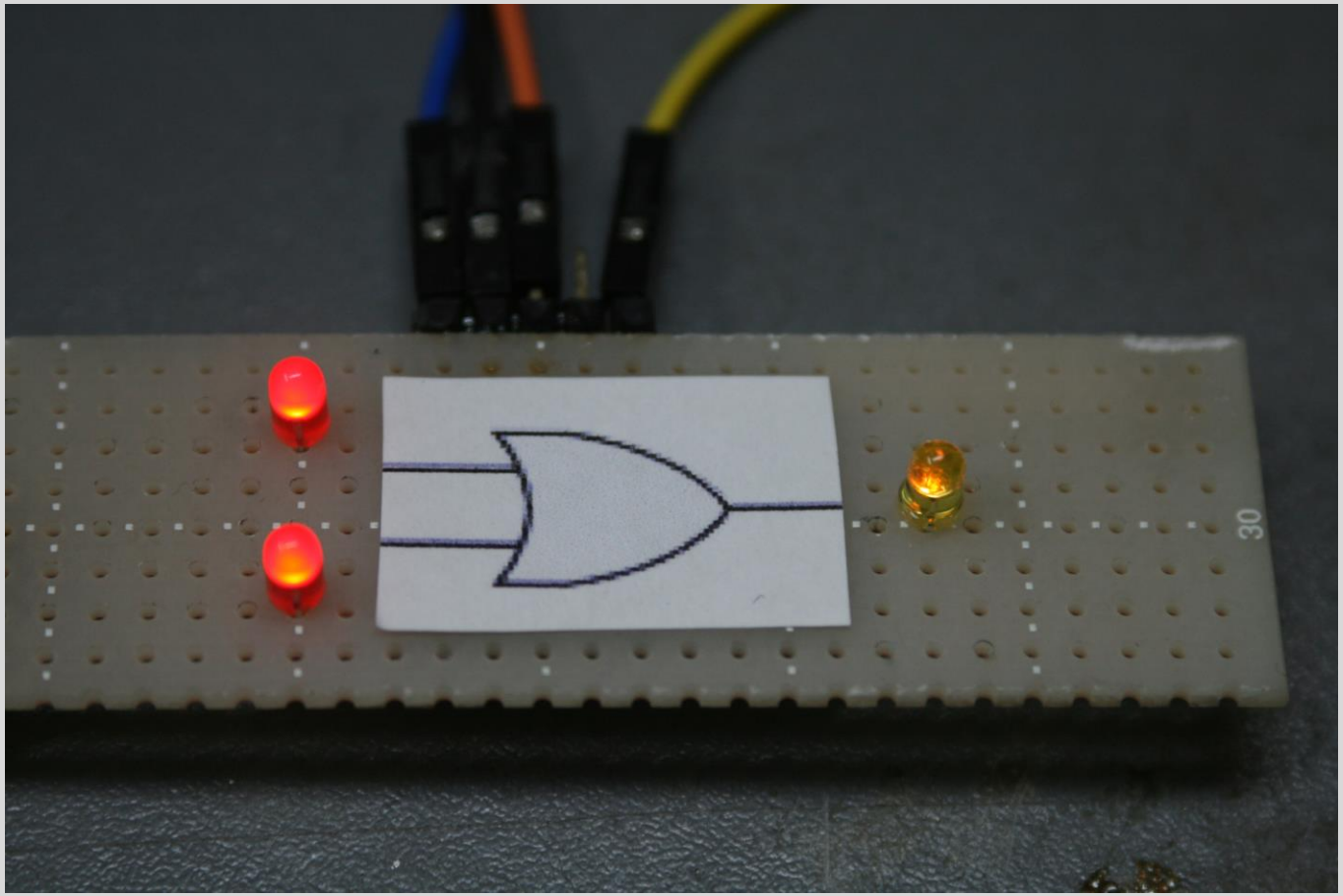
入力が１，１→出力は１



NANDとはNANDぁ(何だ)？

基本の論理「OR」

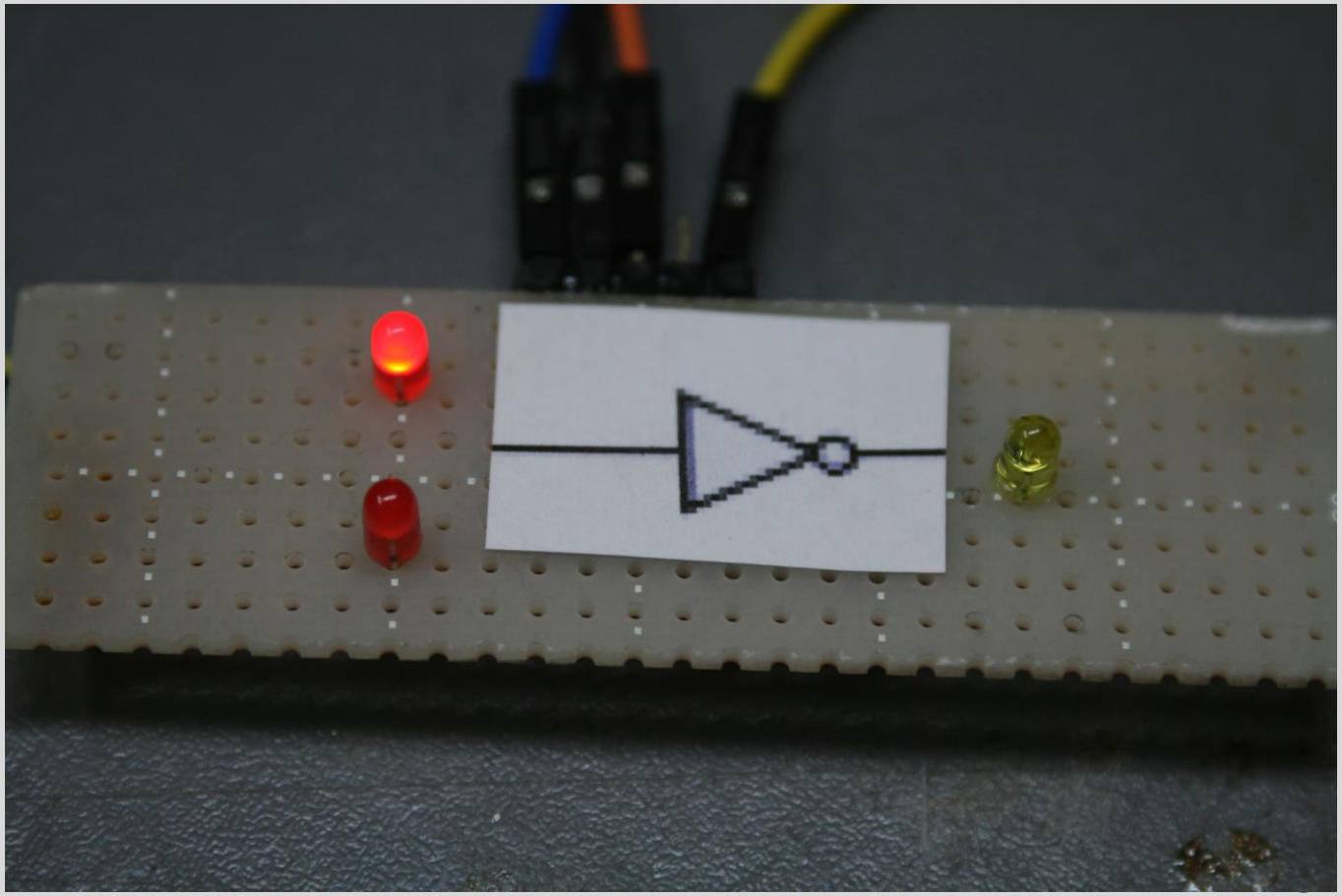
入力が１，１→出力は１



NANDとはNANDぁ(何だ)？

基本の論理「NOT」

入力が１→出力は０



NANDとはNANDぁ(何だ)？

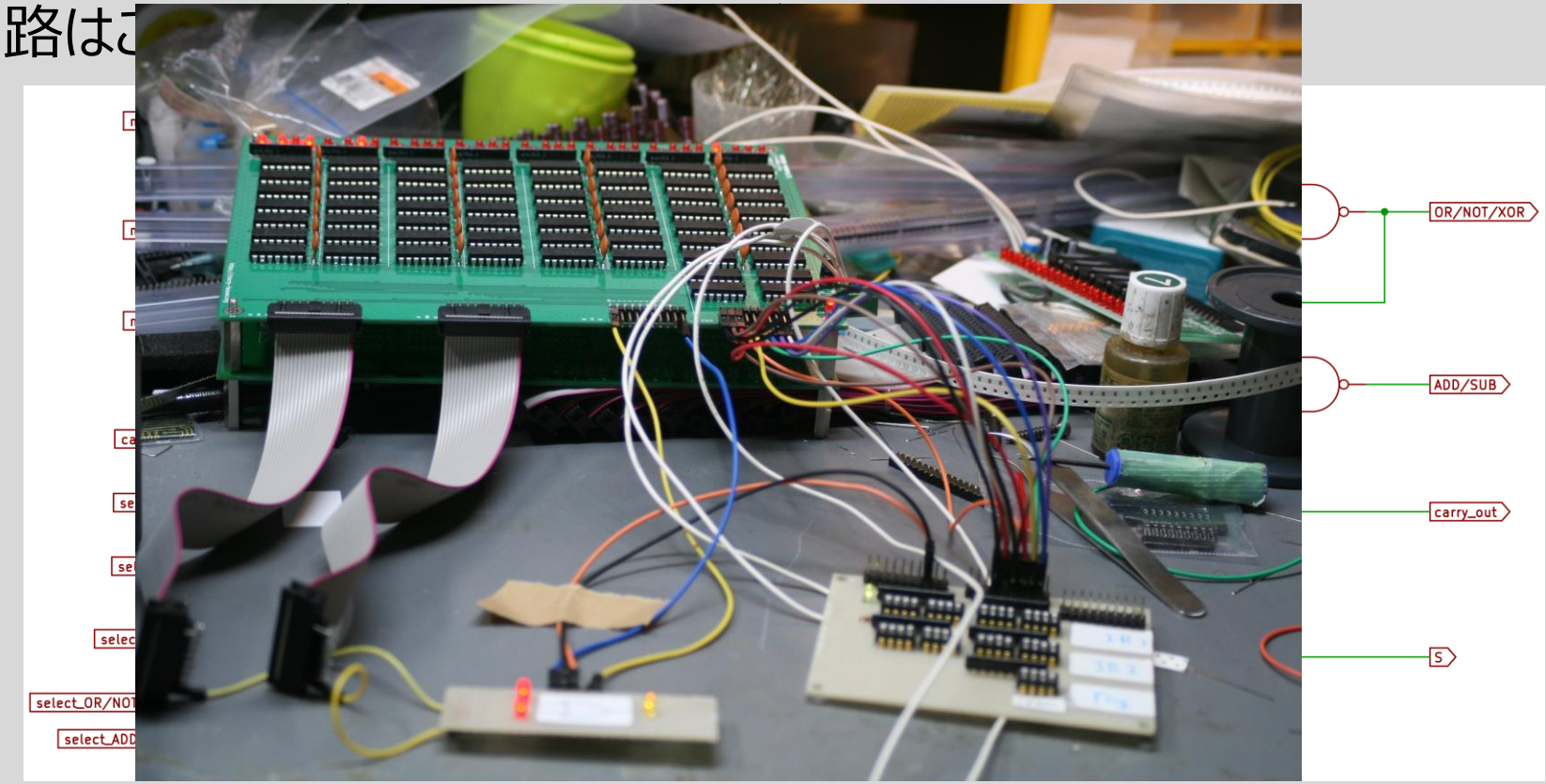
NANDとは基本論理ゲートの「AND」を反転(NOT)した，簡単なルール.

NANDの重要な特性として，NANDのみで他の論理を実現できる.
つまり，**NANDとはなんでも(NANDえ)もできる魔法の素子**である.

実は先ほどの装置もNANDのみで構成されています
NANDのみで何でもできるということを少しは信用してもらえましたか？

NANDとはNANDぁ(何だ)？

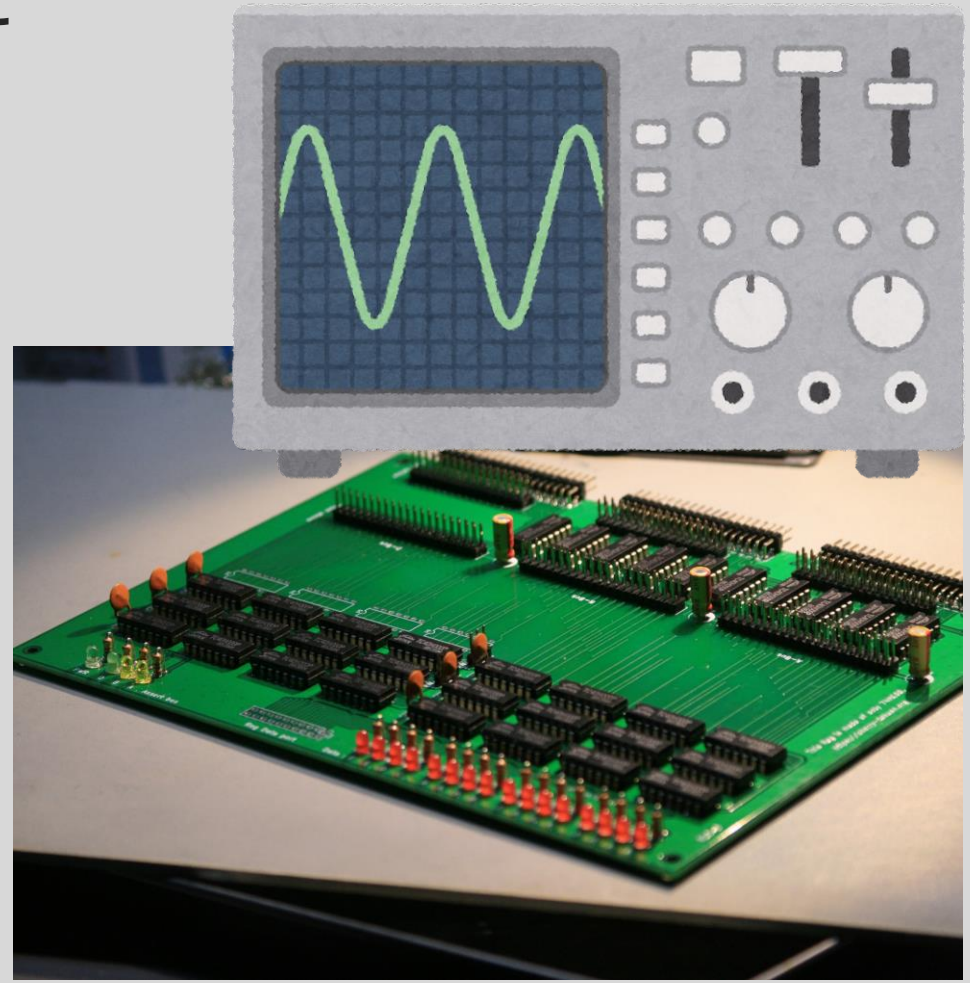
先ほどの回路はこ



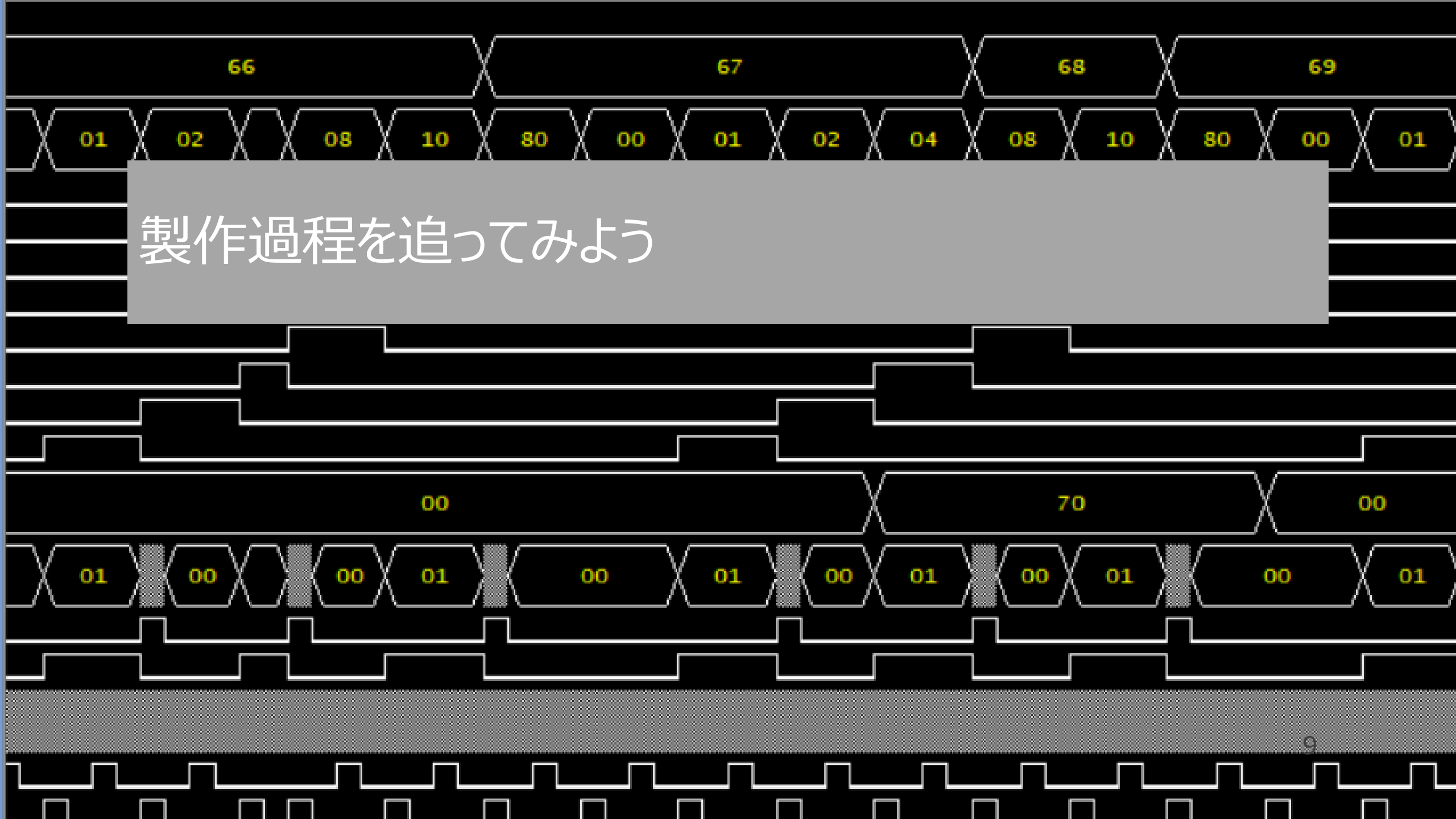
NANDだけでCPUを作るということ

- ICがたくさんでかっこいい
- 回路も見たい信号も全部見ることができる

  →**コンピュータのすべてが自分の手の内にある！！**
- 大きい筐体がピカピカ点滅して見た目がコンピュータ感があってとても良い



いらすとや 「https://www.irasutoya.com/2016/01/blog-post_97.html」


66
67
68
69
01
02
08
10
80
00
01
02
04
08
10
80
00
01
製作過程を追ってみよう
00
70
00
01
00
00
01
00
01
00
01
00
01
00
01

試作と検証(設計製作の初期段階)

回路規模がでかすぎ

仕方ない，回路を工夫して現実的な規模にしよう

大きいゆえに試作がとても重要です

…がっつり設計する前にどのような
る機構を用意できるのかを確認してお

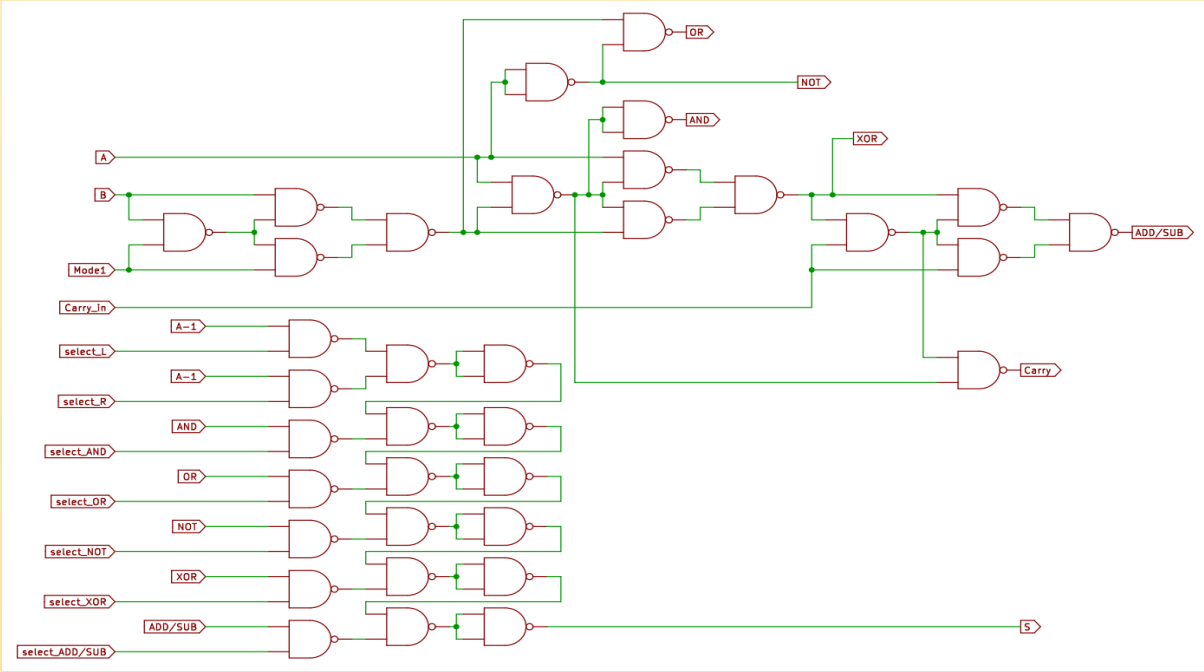
…計
…ので
…いう悲劇に見舞われかねません.

その…動作そのものはもちろん，基板に載
るか，配…は通るか等々…を小さな単位で試作
を繰り返して自分が使えるものを把握しておく
必要があります.

回路を工夫する ～ALU(演算装置)の効率化～

NOTになる箇所を削減する

改善前



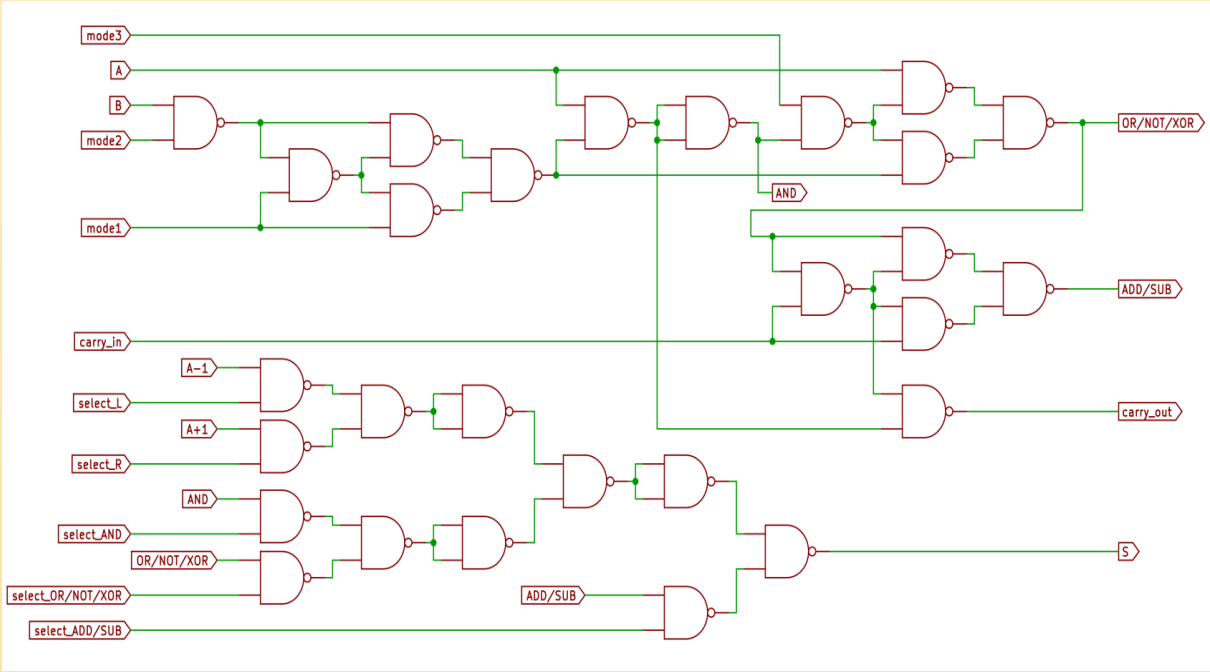
ゲート数：32ゲート(1bit分)
ファンクション数：8
1命令当たりのゲート数(ゲート数/命令数) = 4

NOTとしての利用が多く，
NANDの機能の半分を放棄している状態となっている．

論理圧縮 ～ALU(演算装置)の効率化～

NOTになる箇所を削減する

改善前

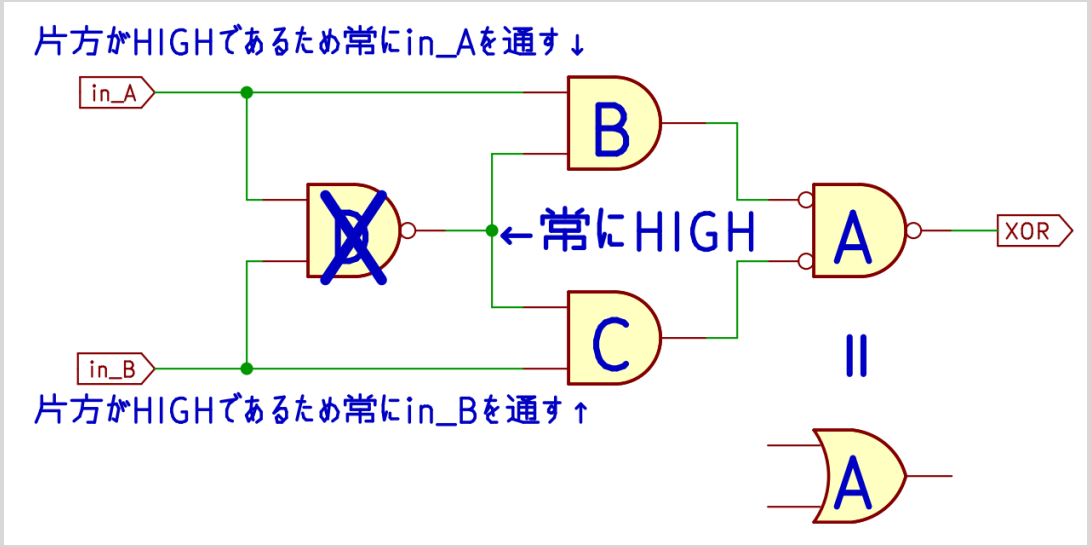


ゲート数：28ゲート(1bit分)
ファンクション数：14
1命令当たりのゲート数(ゲート数/命令数) = 2

NOTになる箇所を極力抑えたところ，同等以上の機能を備えつつむしろコンパクトな回路となった

論理圧縮 ～ALU(演算装置)の効率化～

NOTになる箇所を削減し素子の利用効率を向上する



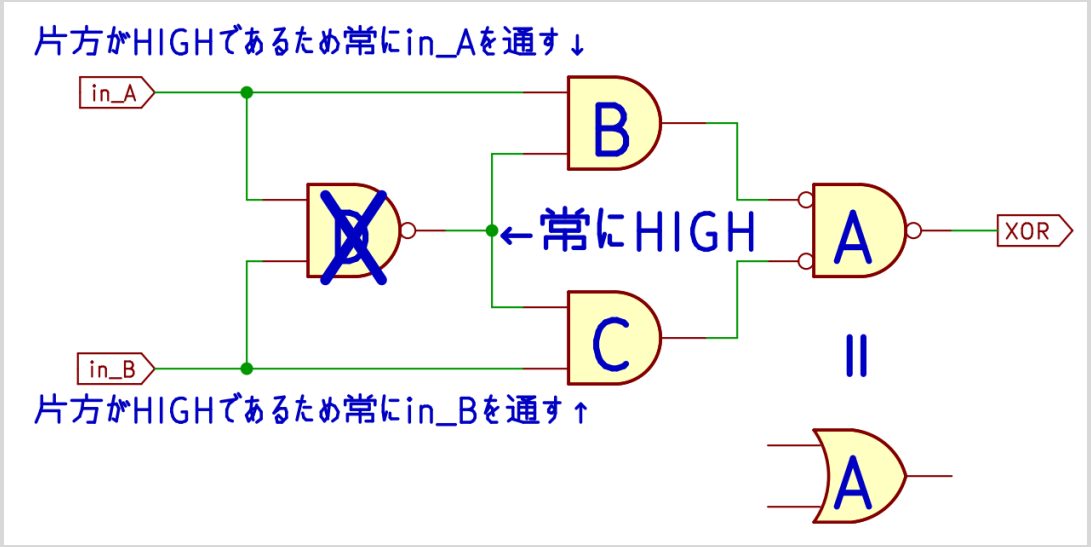
XOR について考える

加算器などでよく出てくるXORの動作を考えてみる.

B,C の NAND のNOT 部分を A に移す
→NAND の A はドモルガン律を利用して OR に変換できる

論理圧縮 ～ALU(演算装置)の効率化～

NOTになる箇所を削減し素子の利用効率を向上する



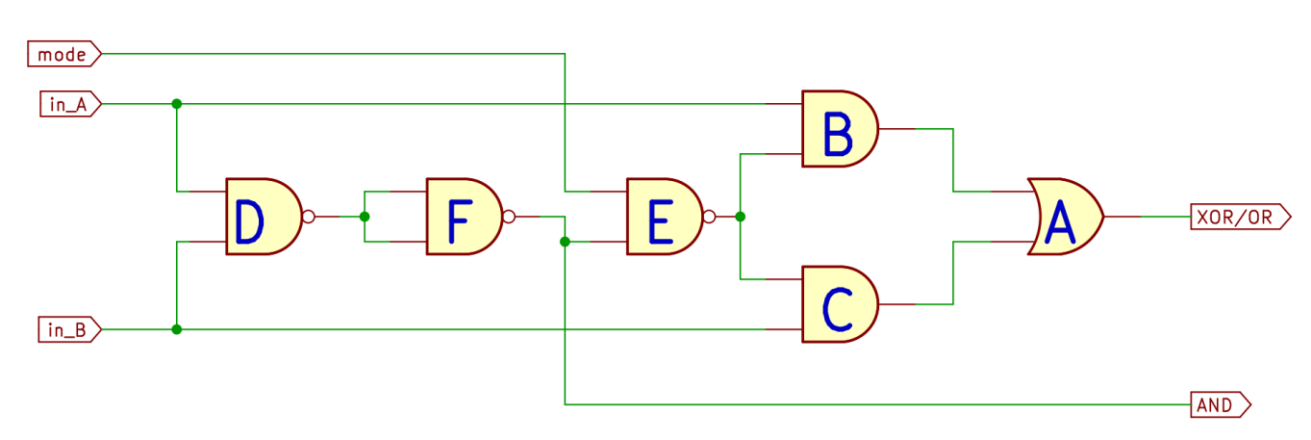
XOR について考える

常に HIGH を出力すると仮定すると
→in A, in B を OR している
ではどのような場合にORをするのか？
→in A,in Bのどちらかまたは両方LOWの時

これらを合わせて入力の内どちらかがLOW
の時にはORをして両方HIGHの時にはLOW

論理圧縮 ～ALU(演算装置)の効率化～

NOTになる箇所を削減し素子の利用効率を向上する

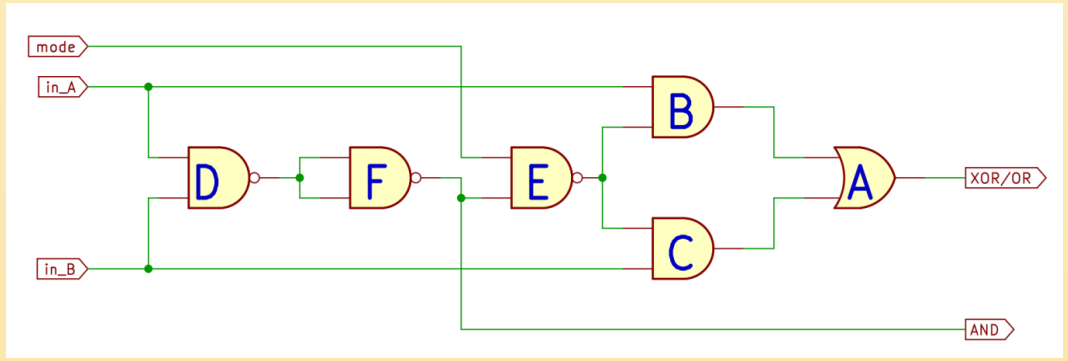


先ほどの「Dの出力をHIGHに固定する」を実現するためにE,Fを追加
するとXORに細工するだけでOR演算が可能になる
加えてFの出力を見てみるとDをNOT, つまり一つの回路でANDも可能になる

論理圧縮 ～ALU(演算装置)の効率化～

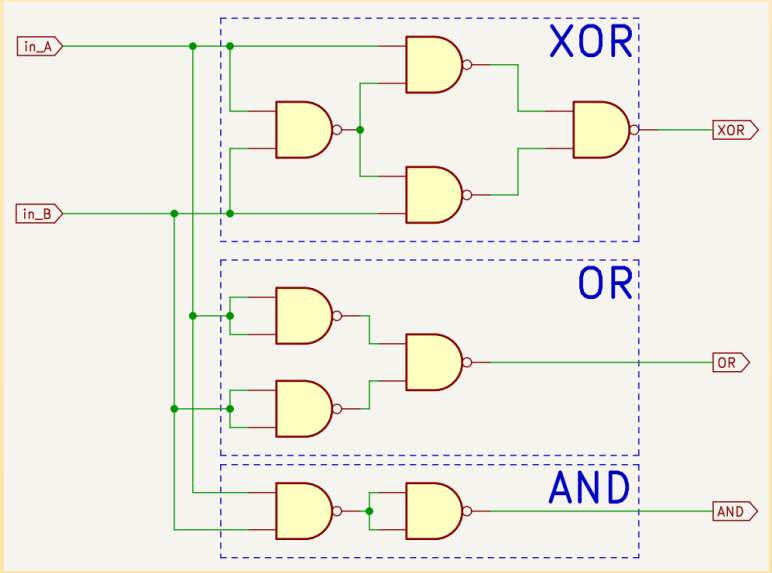
NOTになる箇所を削減し素子の利用効率を向上する

圧縮をした回路



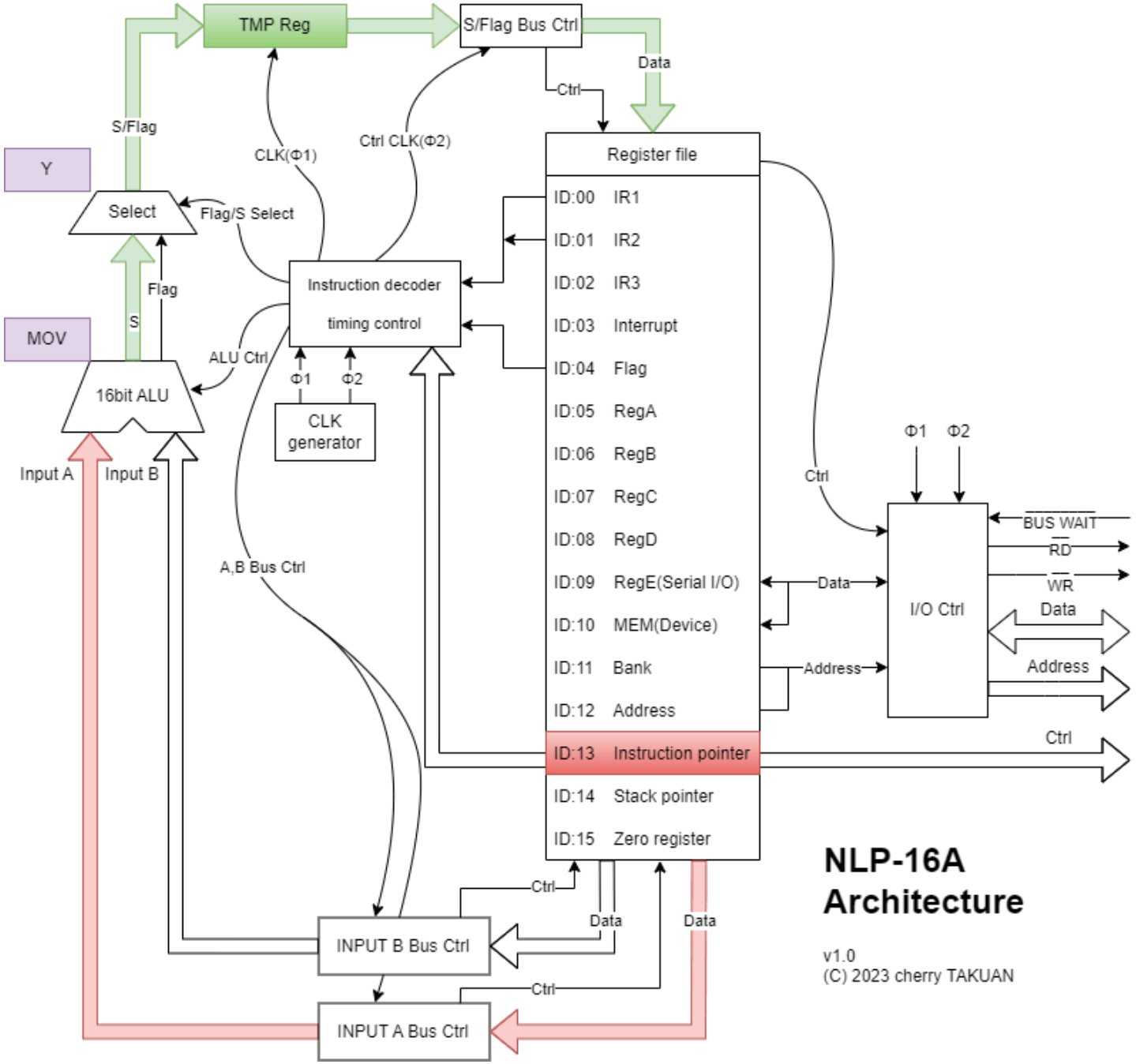
NOT:1か所

特に何も考えずにXOR，OR，ANDをする回路



NOT:3か所

右の回路は大げさではあるが，とりあえずそれぞれの演算をさせて後で選択すればよいという考えではこのようになってしまう．また，選択する回路も入力が2つのほうがゲート数は少ない



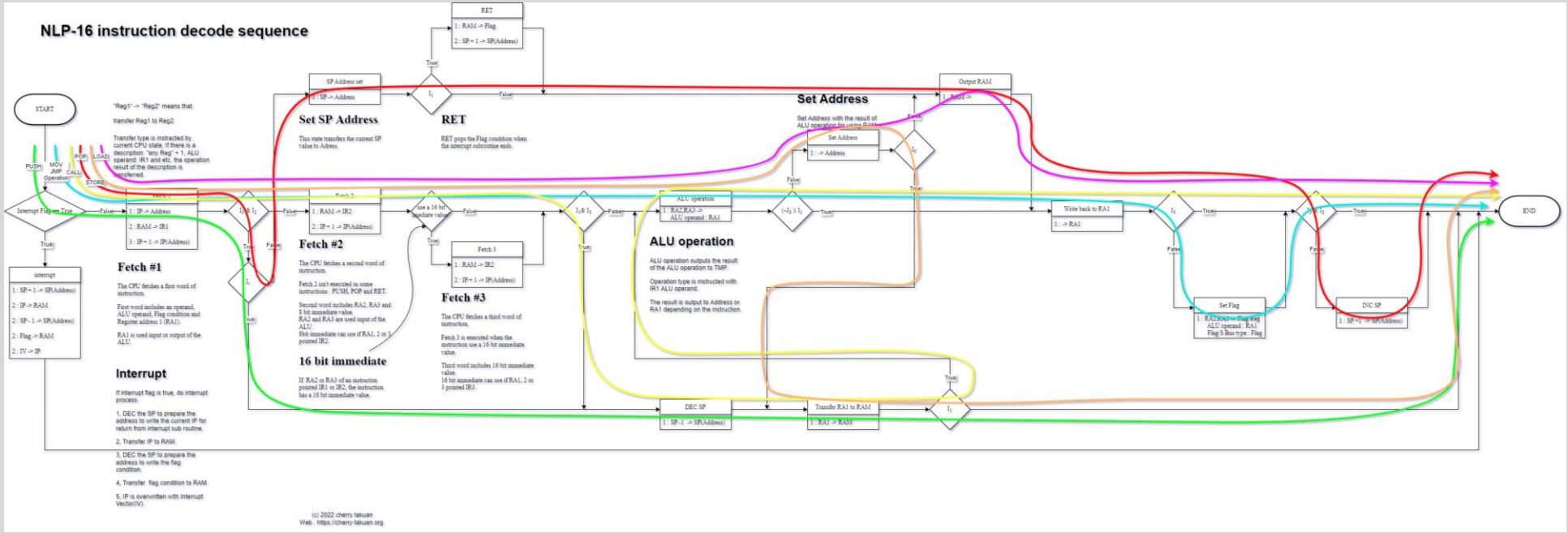
アーキテクチャの設計と検証

各命令実行時の内部動作，データの動きについて詳細を詰めていきます

命令の複雑さを保ったままプロセッサの構造を単純にするためには，データパスを工夫しバス，ALU等をなるべく余すことなく使う必要がある．

左の図は一例として，16bitで表現できる範囲に自己相対アドレッシングでサブルーチンコールを実行している．

アーキテクチャの設計と検証



CPUの各構成要素をどのように繋ぐかが決まれば, どのように命令をデコードするか, どのタイミングでどこに値を移動すれば良いか等のCPUの実際の内部動作を練る

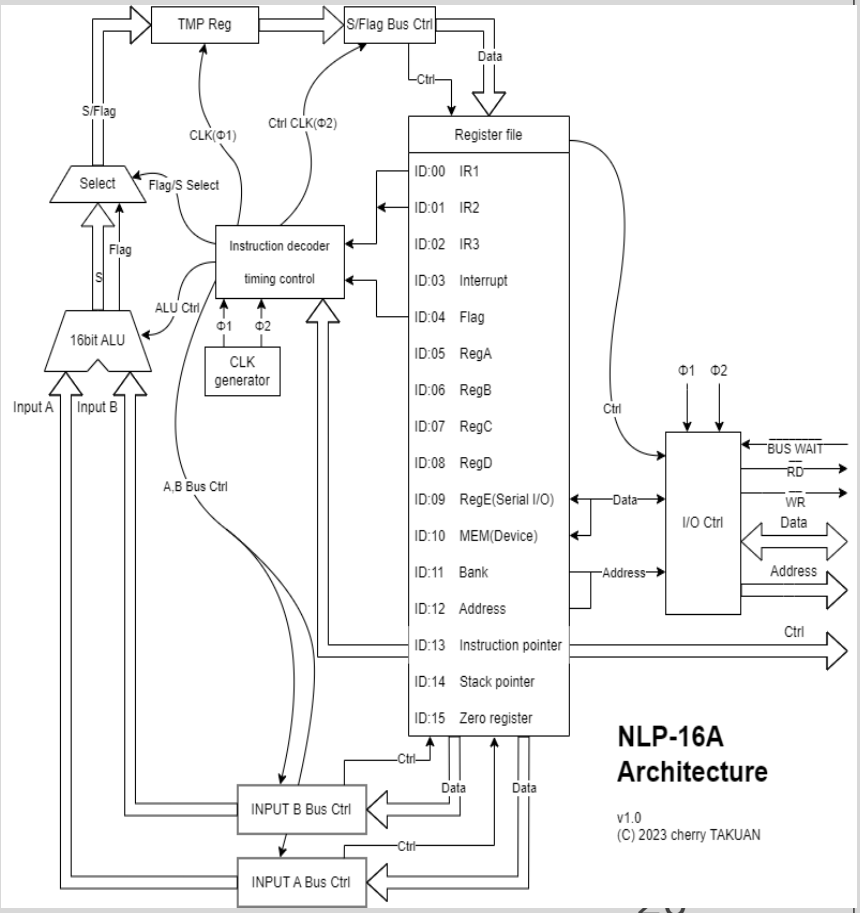
NLP-16,16Aアーキテクチャ色々

前提方針として黎明期のマイクロプロセッサ級の命令を実行可能

- モデルとなるプロセッサには日本の数多くの技術者を育成したTK80搭載MPUであるi8080系を参考

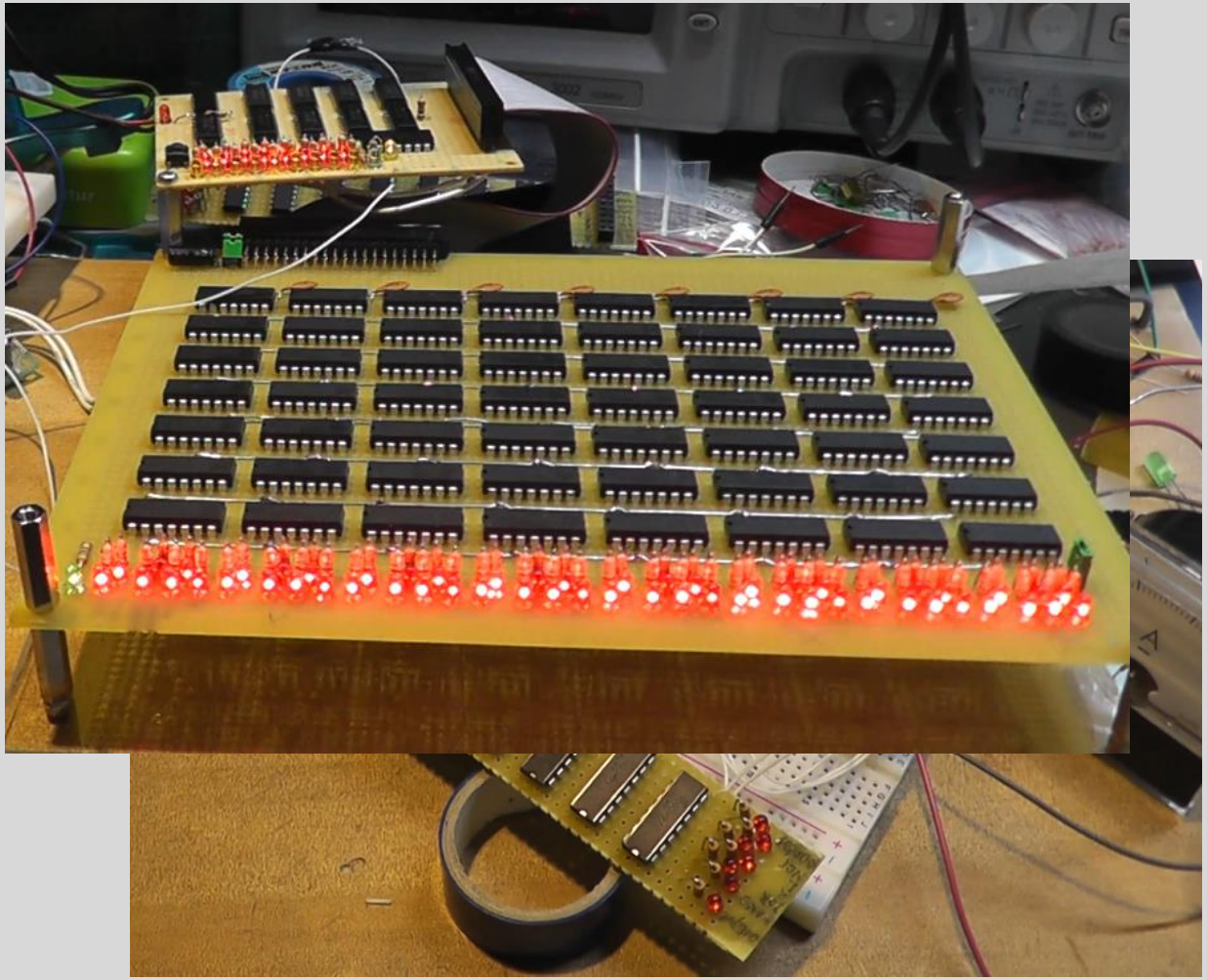
その他最低限のハードウェアに絞る最適化

- ◆レジスタにD-FFではなくD-Latchを用いる
  - ◆通常一度に書き込まれるレジスタは一つのみである.
  - ◆従ってマスタースレーブ型のD-FFのマスターとスレーブを分割し，マスターは1つのみとしても問題がない．(NLP-16ではマスター:TMP,スレーブ:各レジスタとしている)こうすることでレジスタに必要なゲート数が約半分に抑えられる.
- ◆PCや，SPのインクリメント，デクリメントはALUの機能を利用.
  - ◆パラレルインプットのあるカウンタ「161」などを使うことが多いが，NLP-16ではPCやSPは完全に普通のレジスタと同様の機能しかない．演算装置は1ヵ所にまとめてハードウェアの規模を抑え，かつ設計の共通化により作業量削減，動作検証を簡略化
- ◆似た機能の命令は実行時のシーケンスを共通にする
  - ◆JMPと演算命令は演算対象が異なるだけなので共通化する
  - ◆PUSH/CALLはSPを計算し，レジスタ->MEMを行うため共通化できる.またMEMのアドレスをSPではなく任意に設定すればSTOREとも共通化可能．など



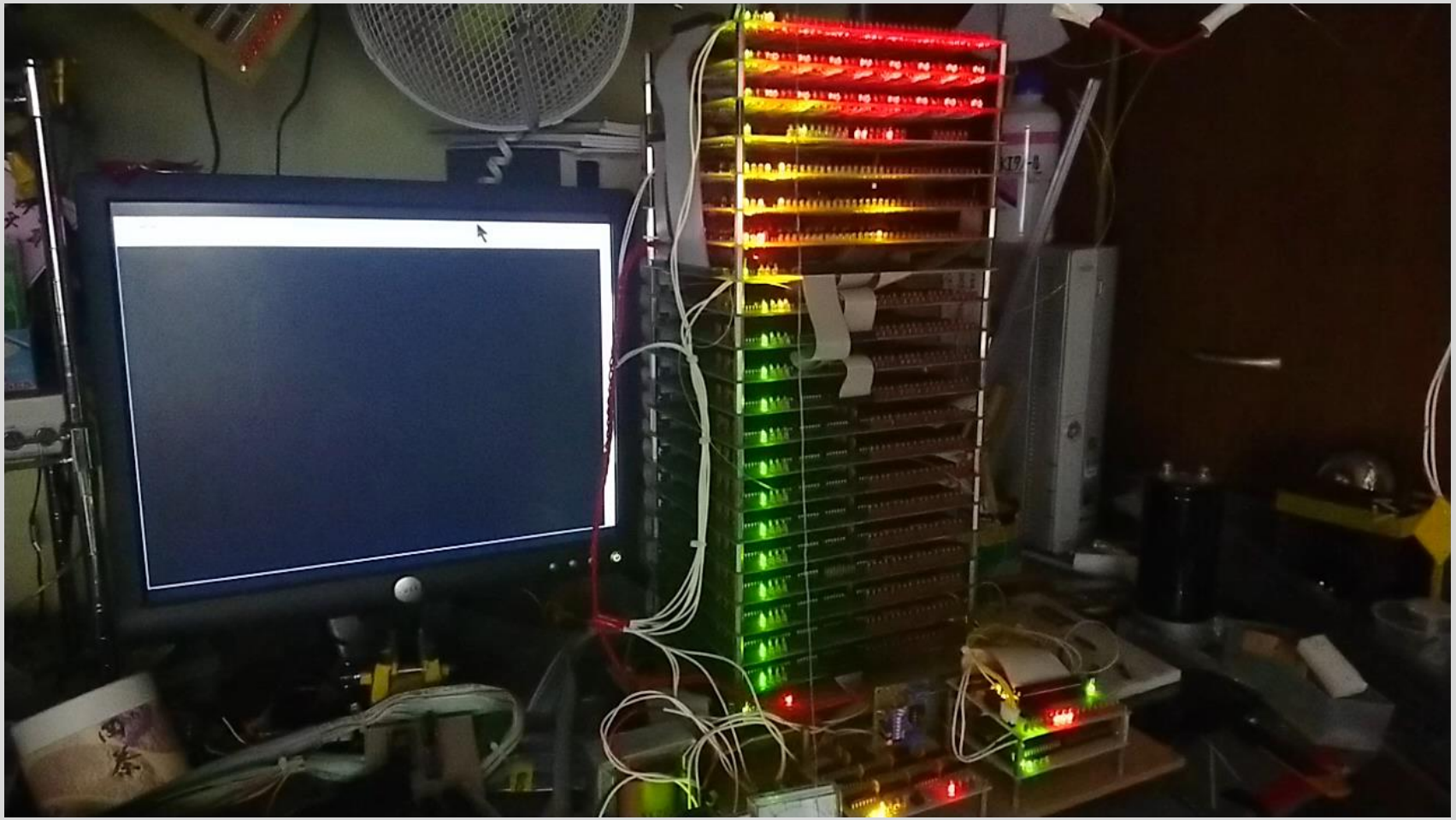
動作の試験環境

- 第一世代
  - 手動で一つ一つテスト
- 第二世代
  - ある回路の1bit分はマスターとして確認済みのものを用意し，そのマスターと他のbitの動作を自動で比較
- 第三世代
  - コンピュータを活用し，試験対象の制御，様々な信号の一括比較を自動で行う



動作の検証環境(第3世代)

最終的にはこうなりました　色々あって調子が悪いのでここでご紹介



ご清聴ありがとうございました

リンク等

- Web：https://cherry-takuan.org
- Twitter：cherry_takuan
- E-mail：cherry@cherry-takuan.org