

順序回路

- 内部状態を保持し、
- 入力と入力前の状態とによって、
- 出力と出力後の状態が決まる

許される内部状態: $Q \subset \{0, 1\}^B$

n 入力 m 出力の順序回路は **Boole** 関数

$$f : Q \times X_f \longrightarrow Q \times \{0, 1\}^m$$

(ここに $X_f \subset \{0, 1\}^n$ は許される入力全体)
を定める

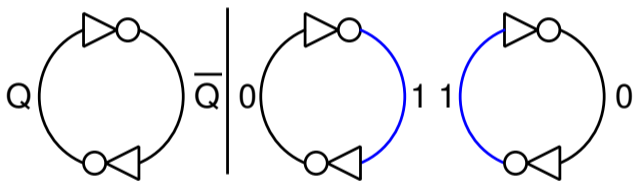
内部状態を計算機内に如何に保持するか

- コンデンサに電荷を蓄える

- 複数の安定状態を持つ論理回路で実現
例: フリップフロップ

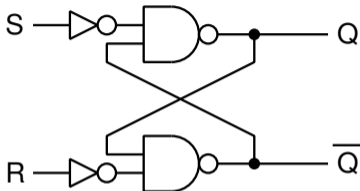
フリップフロップ (flip-flop)

原理図:



これに入出力端子を付ける

SR-フリップフロップ (Set-Reset)

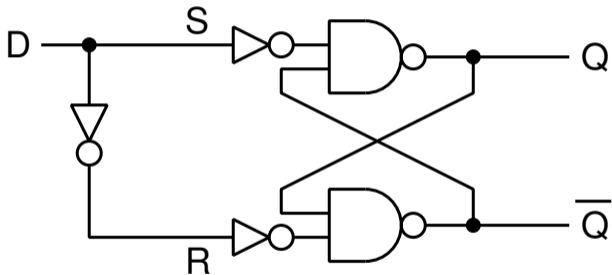


$(S, R) = (1, 1)$ は禁止入力
($X_f = \{(0, 0), (0, 1), (1, 0)\}$)

S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	x

入力が $(S, R) = (1, 1)$ とならない回路設計

→ $S = \bar{R}$ となるようにしてみる

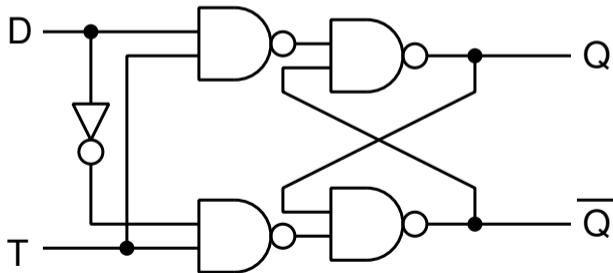


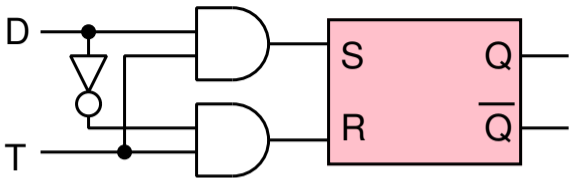
→ 入力 D を保持・出力

これだと D をそのまま流しているのと同じ
→ 書込スイッチを付けよう

T : スイッチ入力

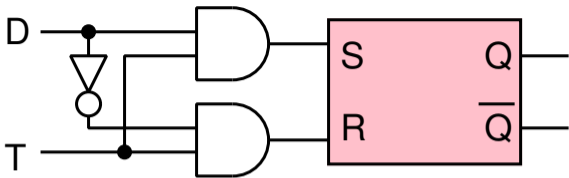
- $T = 0$: そのまま
- $T = 1$: D を取り込む





T	D	S	R	Q
0	0	0	0	Q
0	1	0	0	Q
1	0	0	1	0
1	1	1	0	1

T	D	S	R	Q
0	*	0	0	Q
1	*	D	\overline{D}	D



- Q からの出力により他の値を計算
- 他からの入力により D を計算

→ Q が D に影響を与えることにより、
状態が変わってしまう

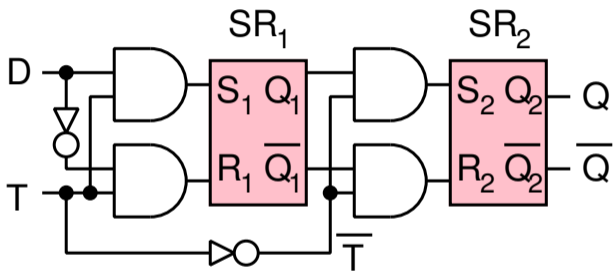
回路の他の部分と

タイミングを合わせる必要あり

→ 出力を一旦せき止めて、
一段階づつ計算を進める

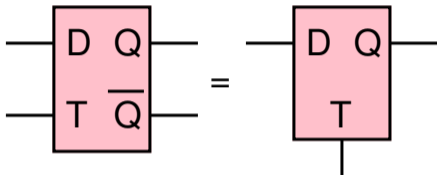
→ **D-フリップフロップ・クロックパルス**
の利用

D-フリップフロップ (Double, Delay)



- $T = 1, \bar{T} = 0 \implies \mathbf{SR}_1$: 取込、 \mathbf{SR}_2 : 保持
- $T = 0, \bar{T} = 1 \implies \mathbf{SR}_1$: 保持、 \mathbf{SR}_2 : 取込

D-フリップフロップ (Double, Delay)

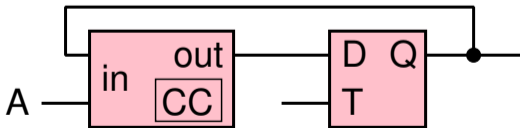


- $T = 1$: 入力 D を内部に取り込む
(出力 Q は変わらない)
- $T = 0$: 内部状態を Q に送り出す
(入力 D は受け付けない)

例: (CC は適当な組合せ回路)

- $T = 1$: 入力 D を内部に取り込む
- $T = 0$: Q に出力 $\rightarrow CC$ の入力へ
 $\rightarrow CC$ での計算結果が D に到達
- $T = 1$: 新たな D を内部に取り込む

$\rightarrow T = 1 \rightarrow 0 \rightarrow 1$ と変化する度に、
計算が一段階進む



一定の周期で $1 \rightarrow 0 \rightarrow 1$ を繰り返す信号

(クロックパルス, **clock pulse**) を

水晶発振器などで発生させることにより、

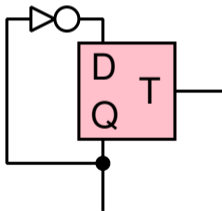
回路全体での同期を取って、計算を進める。

クロックパルスの周波数

→ 計算機の動作の速さ
(動作 **clock** **Hz**)

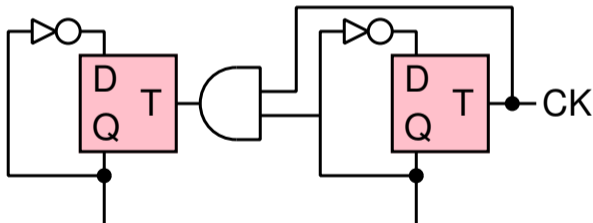
(二進) カウンタ

T が一周期変化する度に
内部状態が $1 \rightarrow 0 \rightarrow 1$ と変化する。



四進カウンタ

T が一周期変化する度に、内部状態が
 $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ と変化する。



計算機を構成する基本的な論理回路の
部品については紹介した。

(ここまで復習)



どのような部品をどう組み合わせれば
計算機の機能を実現できるか

計算機が実現すべきこと

- データの保持・書込・読出 → 順序回路
- データの処理 (演算) → 組合せ回路
- (データの入出力)
- 実行の制御
 - ★ プログラムの読出
 - ★ 順次実行
 - ★ 条件分岐

データの保持・書込・読出

データの保持: 記憶装置 (memory)

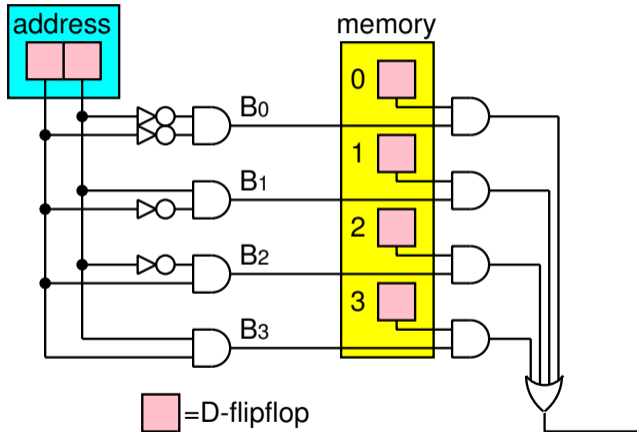
D-フリップフロップを必要なだけ並べる

その中の

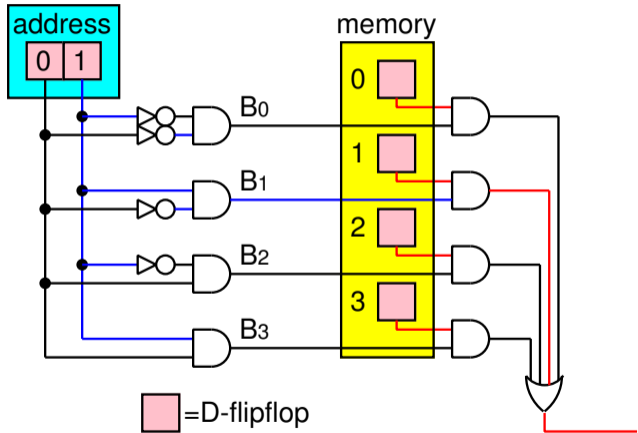
- どれを読み出すか
- どれに書き込むか

→ 番地 (address) で管理

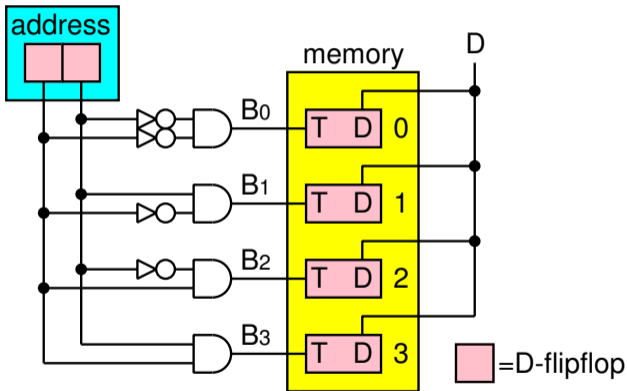
データの読出し



データの読出し: 1番地のデータを読出し



データの書込み



データの保持・書込・読出

実際には、何 bit かで一まとまりとして、

データの読み書き・演算等の処理を行なう

→ 1 語 (**word**)

(例: 1 **word** = 8 **bit**)

⇒ 8 つを並列に並べておく)

→ **bit CPU**

(一度に処理できるデータの大きさを表す)

プログラム内蔵方式 (von Neumann 型)

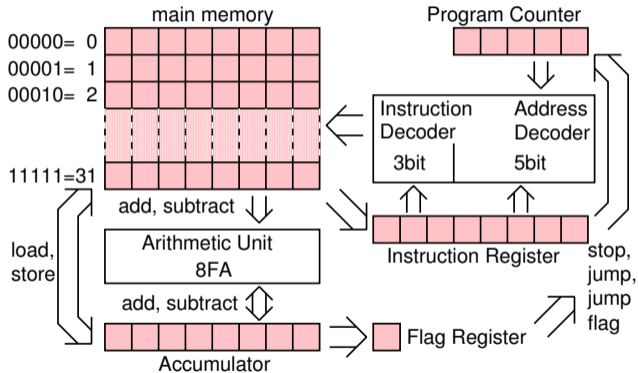
プログラム・データを共にメモリ上に置く

→ 主記憶装置

実行の流れ: 以下を繰り返す

- プログラムを一命令ずつ読み出す
- 命令の実行

説明用の簡易モデル



プログラム内蔵方式 (von Neumann 型)

プログラム・データを共にメモリ上に置く

→ 主記憶装置

実行の流れ: 以下を繰り返す

- プログラムを一命令ずつ読み出す
- 命令の実行

次にどの命令を読み出すか

→ プログラムカウンタ

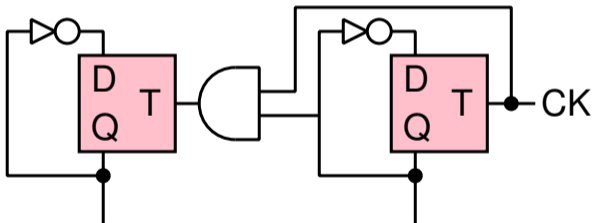
プログラムカウンタ (program counter)

次に読み出す命令の番地 (address) を
保持する一時記憶場所 (register)

- 順次実行 (通常):
次の番地
→ 基本はカウンタで実現
- 実行制御 (無条件ジャンプ・条件分岐):
指定の番地
→ 命令による値の書き換え

四進カウンタ

T が一周期変化する度に、内部状態が
 $00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ と変化する。



命令レジスタ (instruction register)

読み出した命令の一時記憶場所が必要

→ 命令レジスタ

命令の種類:

- 演算 (加減乗除・桁ずらし他)
- 演算対象の指定
- 演算結果の保存
- 実行制御
- 終了

など

演算の実行の実際

実際には、二項演算は次のように行なう。

例: $C \leftarrow A + B$

- (1) A を読み出す
- (2) B を (読んで) 足し込む
- (3) C に書き込む

読み出した値・演算結果の一時記憶場所が必要

→ **アキュムレータ (accumulator)**

命令の形式: 命令の種類 + 番地

命令の種類:

- 主記憶からアキュムレータへの読出し (load)
- アキュムレータから主記憶への書込み (store)
- 演算 (add, subtract)
- 実行制御 (jump, jump flag)
条件分岐の判断 → フラグレジスタ
- 終了 (stop)

→ これらを論理回路で実装すれば良い

必要な構成要素

- 主記憶装置
- プログラムカウンタ
- 命令レジスタ
- アキュムレータ (汎用レジスタ)
- フラグレジスタ

- 番地解読回路・命令解読回路
- 演算回路 (補助演算回路)

- パルス発生器

説明用の簡易モデルの仕様

- **1 word = 8 bit (= 1 byte)**
 - 一度に扱うデータの大きさを規定
 - ★ アキュムレータ
 - ★ 命令レジスタ
 - ★ 演算回路
 - ★ 主記憶 1 番地分
- **命令部: 3 bit、番地部: 5 bit**
 - 扱える命令・番地の個数を規定 (制約)

説明用の簡易モデルの仕様

- **命令部: 3 bit、番地部: 5 bit**
 - ★ **命令:** $2^3 = 8$ つ以内
load, store, add, subtract,
jump, jump flag, stop.
 - ★ **番地:** 0 番地から 31 番地まで
 - **主記憶** $2^5 = 32$ **byte**
 - **プログラムカウンタ**は **5 bit**

説明用の簡易モデルの仕様

