計算機がすべきこと

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

データの保持: 記憶装置 (memory)

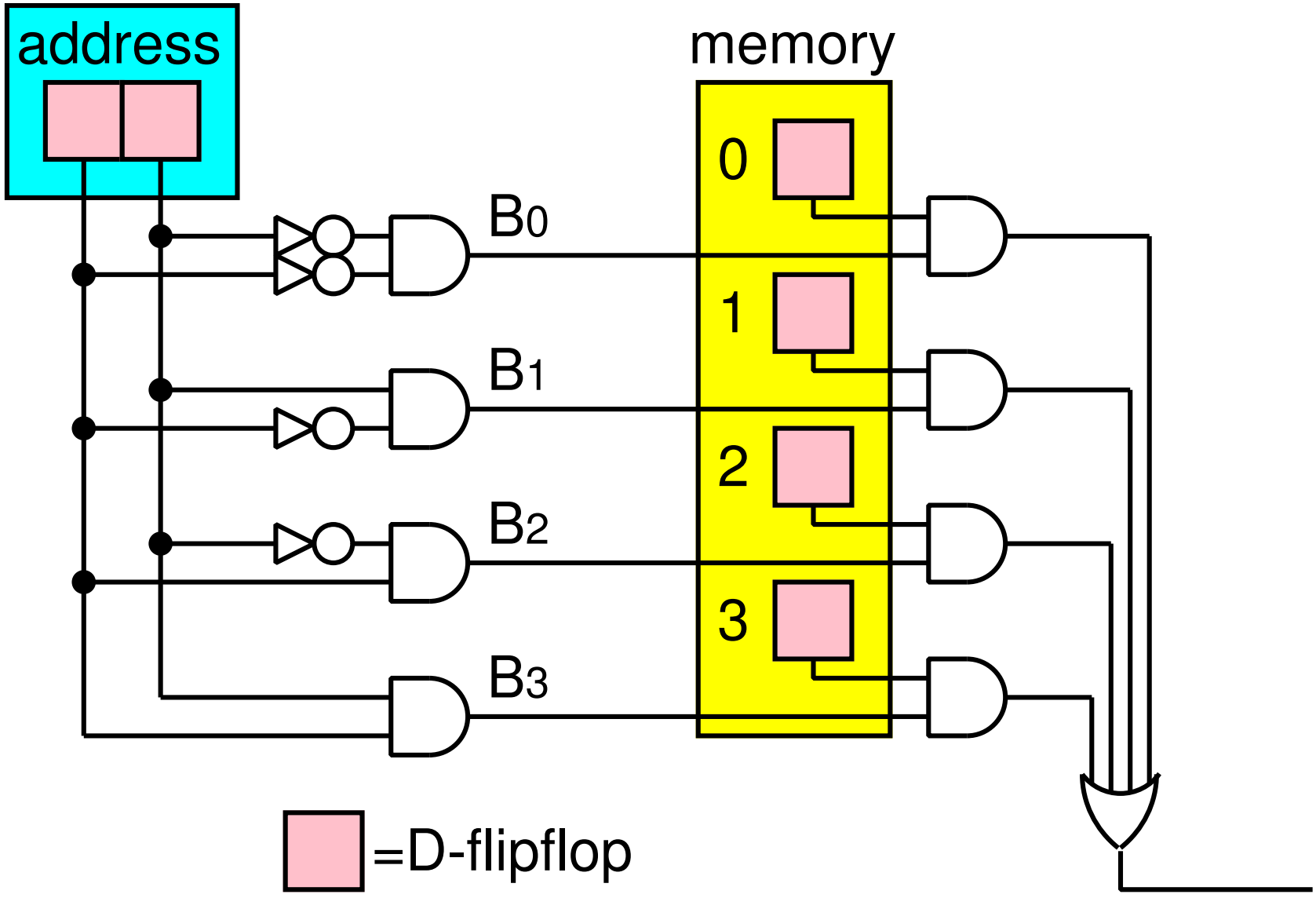
D-フリップフロップを必要なだけ並べる

その中の

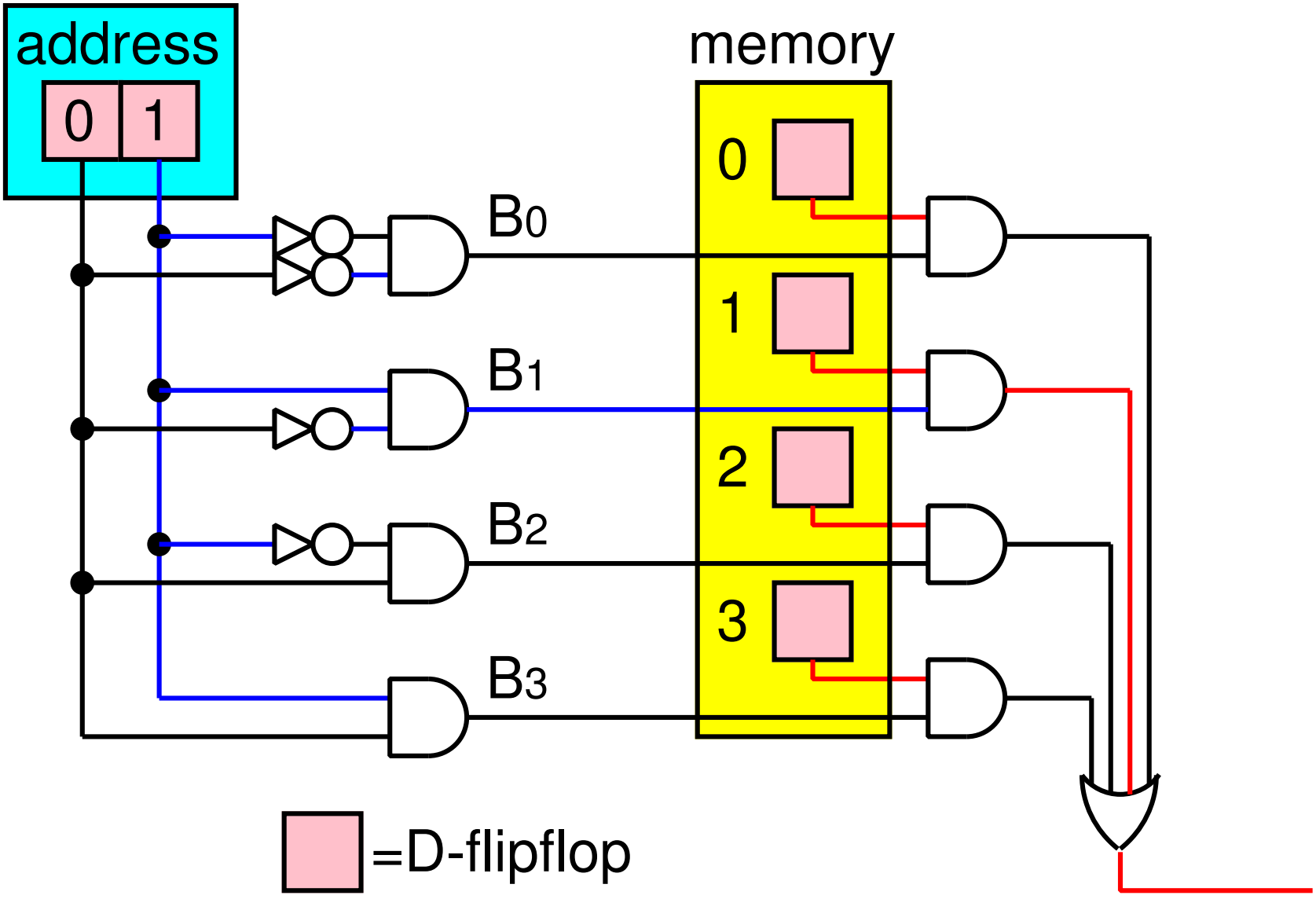
- どれを読み出すか
- どれに書き込むか

$\longrightarrow$ 番地 (address) で管理

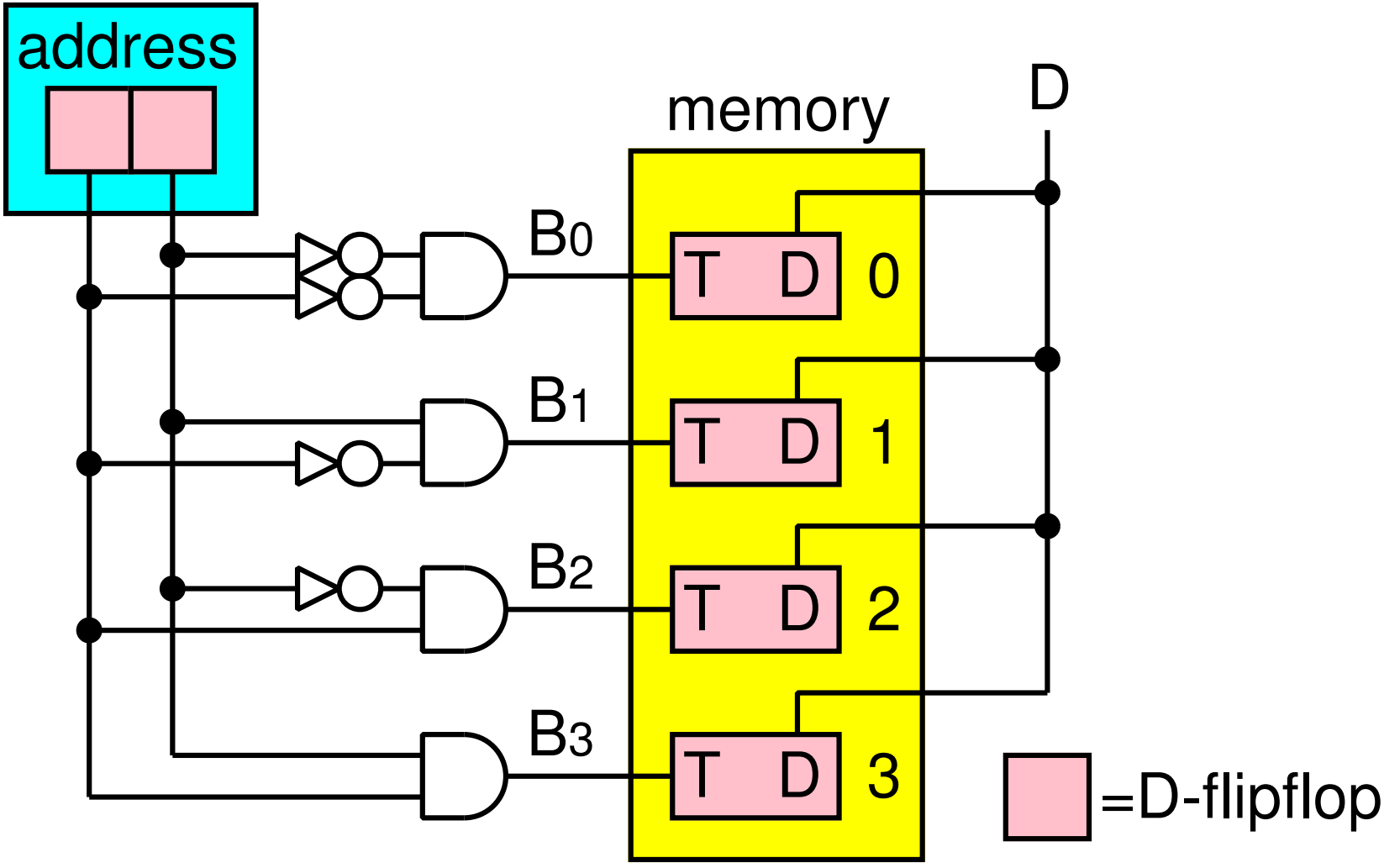
データの読出し:



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:



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

$\longrightarrow$ 1 **語 (word)**

(例: 1 **word** $=$ 8 **bit**

$\Longrightarrow$ 8 **つを並列に並べておく)**

$\longrightarrow$ ○○ **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

$\longrightarrow$ **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づつ読み出す
- 命令の実行

次にどの命令を読み出すか

$\longrightarrow$ **プログラムカウンタ**

プログラムカウンタ

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

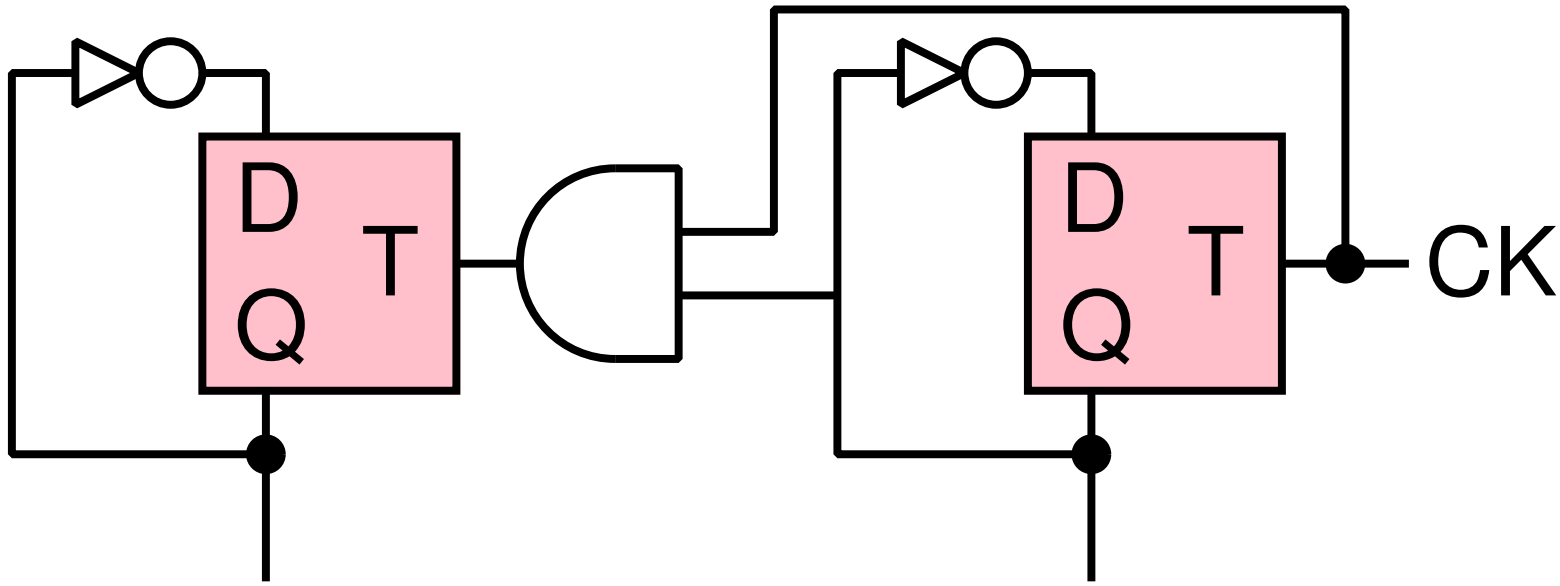
- **順次実行 (通常):**
  次の番地
  $\longrightarrow$ 基本はカウンタで実現

- **実行制御 (無条件ジャンプ・条件分岐):**
  指定の番地
  $\longrightarrow$ 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。**



読み出した命令の一時記憶場所が必要

$\longrightarrow$ **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

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

$\longrightarrow$ **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
  $\longrightarrow$ **フラグ (flag) 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:

- 1 **word** $= 8$ **bit**$(= 1$ **byte**$)$
  $\longrightarrow$ **一度に扱うデータの大きさ**
  - $\star$ **アキュムレータ**
  - $\star$ **命令レジスタ**
  - $\star$ **演算回路**

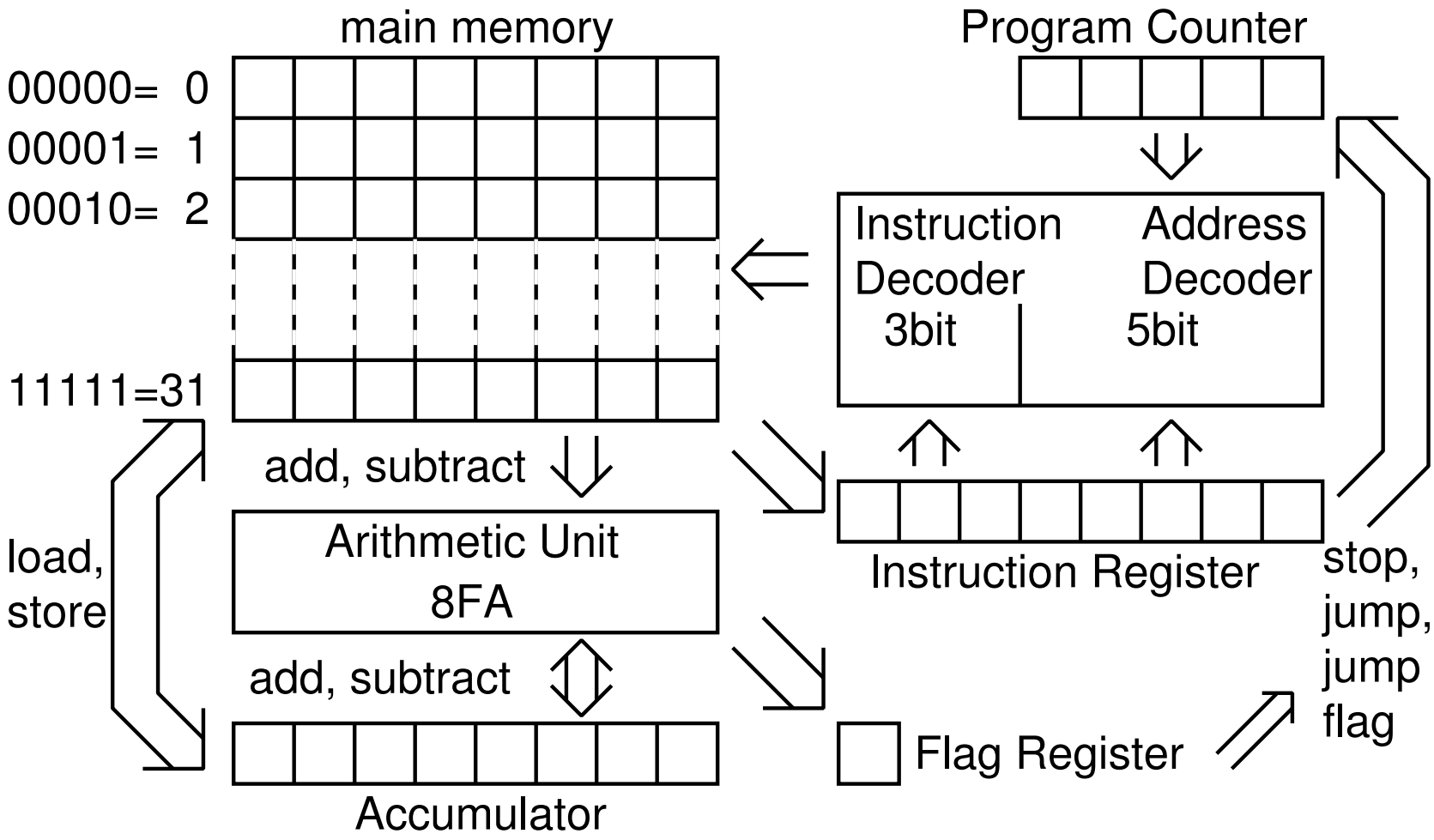
- **命令部:** 3 **bit、番地部:** 5 **bit**

- **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**
    $\longrightarrow$ **主記憶 $2^5 = 32$ byte**
    $\longrightarrow$ **プログラムカウンタは 5 bit**

説明用の簡易モデル:

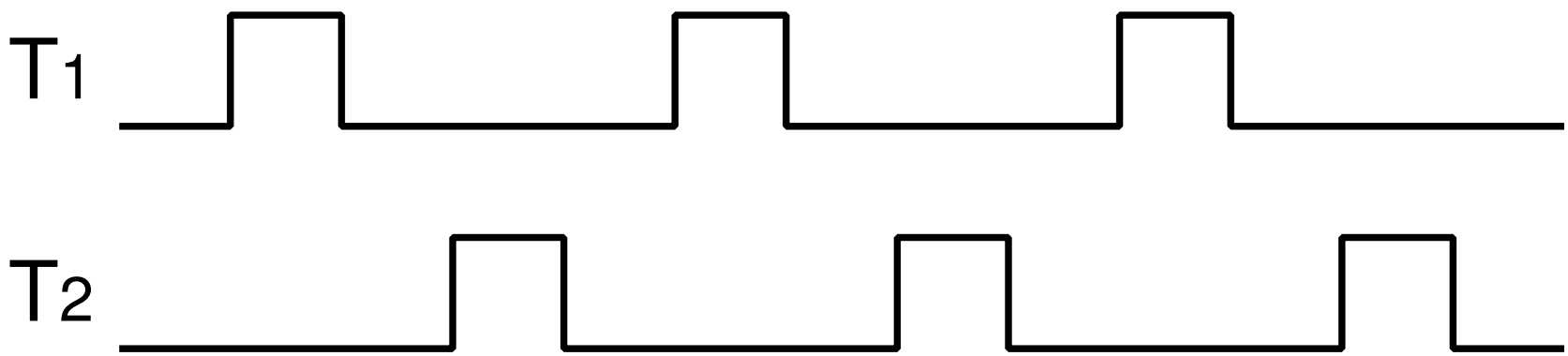

main memory
Program Counter
00000= 0
00001= 1
00010= 2
11111=31
Instruction Decoder 3bit
Address Decoder 5bit
add, subtract
load, store
Arithmetic Unit 8FA
add, subtract
Instruction Register
stop, jump, jump flag
Flag Register
Accumulator

説明用の簡易モデル:

パルス発生器:

交互に 1 となる 2 つのパルス T_1, T_2 を発生

- $T_1 = 1$ **で次の命令の読出し**
  プログラムカウンタを 1 進める
- $T_2 = 1$ **でその命令の実行**



説明用の簡易モデル:

処理の流れ

$T_1 = 1$:

- PC の値が指定する番地の主記憶の内容を読出して、命令レジスタに書込み
- PC の値を 1 増やす

$T_2 = 1$:

- 命令レジスタの内容の解析と実行
  (命令に応じた所定のレジスタへの書込み)

説明用の簡易モデル:

命令の実行:

- `load`:
  命令レジスタの番地部が指定する番地の主記憶の内容を読出して、Acc に書込み、フラグをセット
  (ここでは符号 bit をコピー)

- `store`: **Acc の内容を読出して、命令レジスタの番地部が指定する番地の主記憶に書込み**

説明用の簡易モデル:

命令の実行:

- `add` ／ `subtract`:
  命令レジスタの番地部が指定する番地の主記憶の内容と Acc の内容とを読出して、演算回路で加算／減算し、Acc に書込み、フラグをセット (ここでは符号 bit をコピー)

  (工夫すると回路は共通化出来る)

説明用の簡易モデル:

命令の実行:

- `jump`:
  命令レジスタの番地部を PC に書込み
- `jump flag`:
  フラグレジスタが 1 のときのみ、命令レジスタの番地部を PC に書込み
- `stop`:
  命令レジスタの番地部を PC に書込み、パルスを止める

(ここで書換えた PC の値が、次の命令読出しで使われる $\longrightarrow$ 実行順番の変更)

説明用の簡易モデル:

命令の実行 (＝ 「計算」):

レジスタまたは主記憶の
現在の値 (状態) に従って、

その値を変更 (書込) すること

各命令がどこへの書込を引き起こすか

- パルス T_1: 命令レジスタ・PC
- `load`: Acc・Flag
- `store`: 主記憶の所定番地
- `add`・`subtract`: Acc・Flag
- `jump`・`jump flag`: PC
- `stop`: PC・パルス発生器