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)

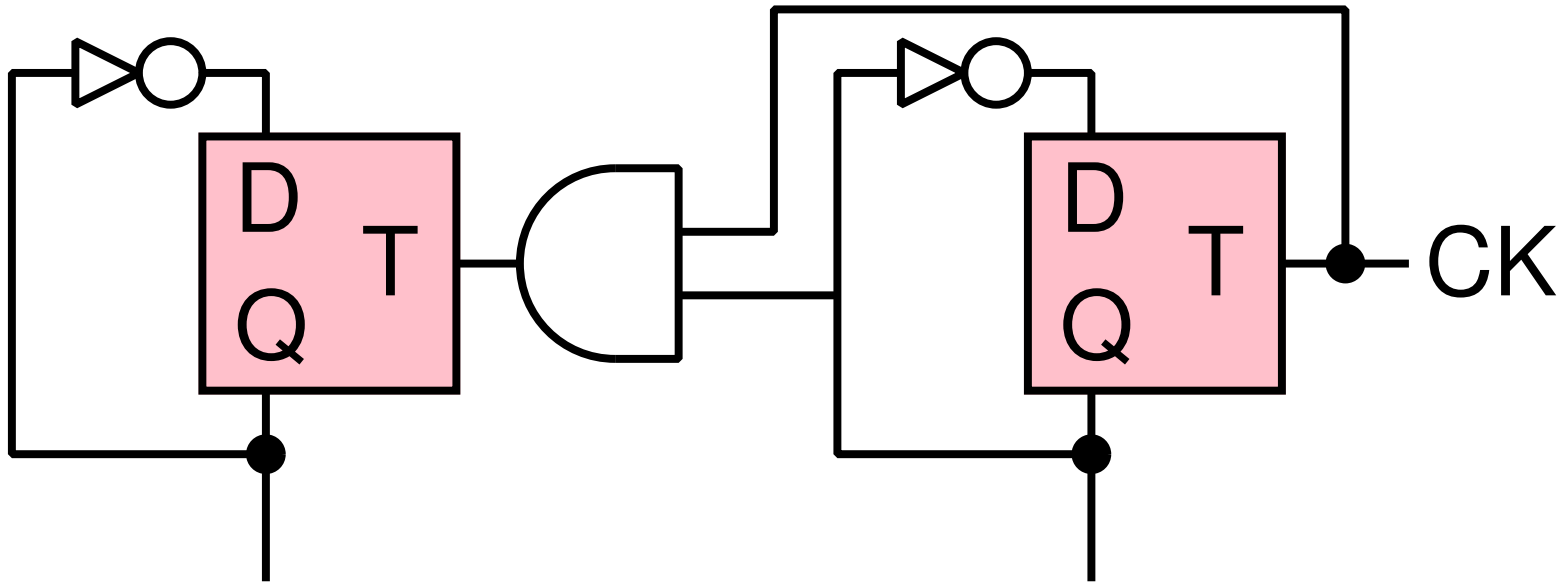
- **順次実行 (通常):**
  次の番地
  $\longrightarrow$ 基本はカウンタで実現

- **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

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

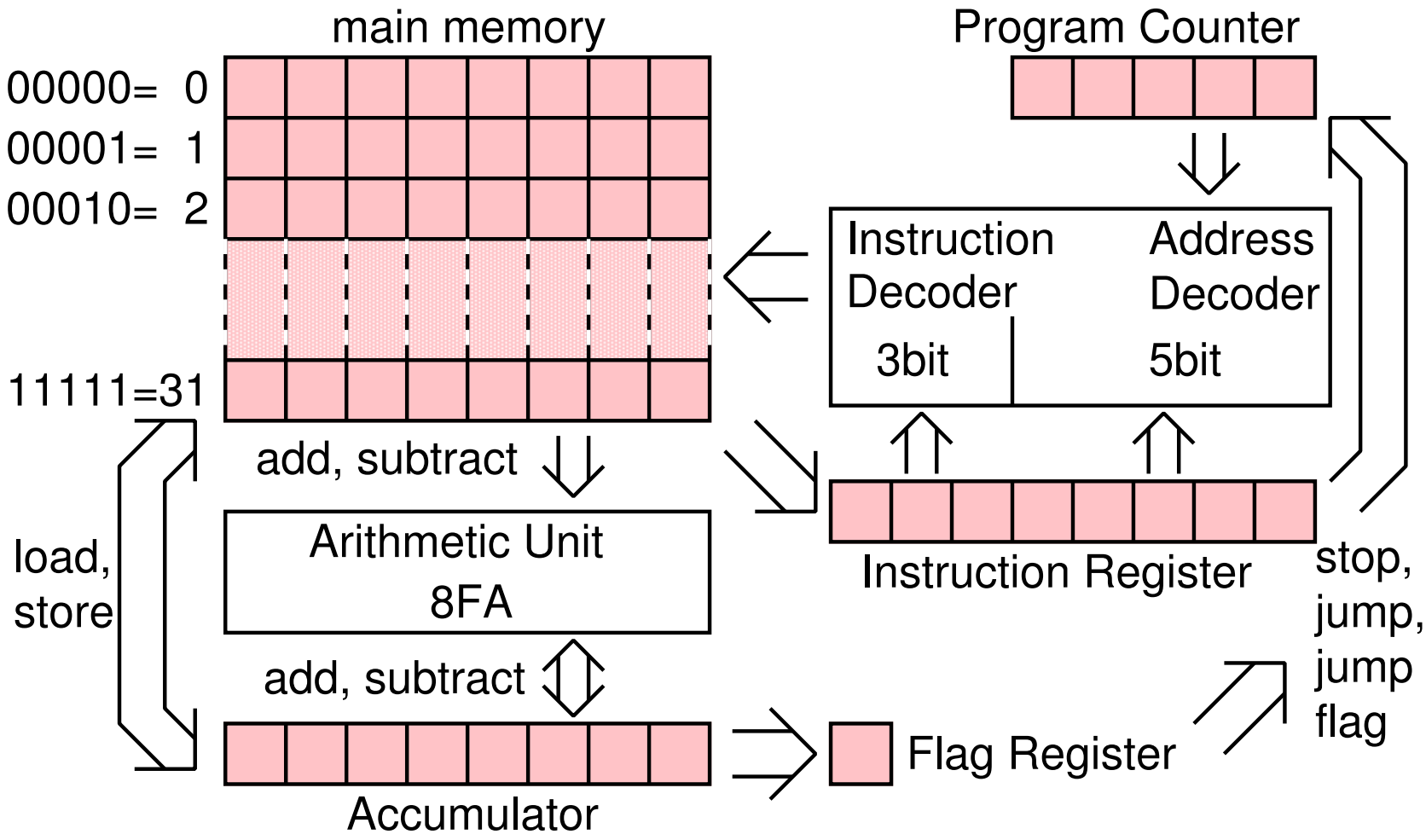
- 命令部: 3 bit、番地部: 5 bit

- **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.
    $\longrightarrow$ **二進 3 桁の番号で扱う**
    $\cdots$ **命令の符号化 (encoding)**

  - ⋆ **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:

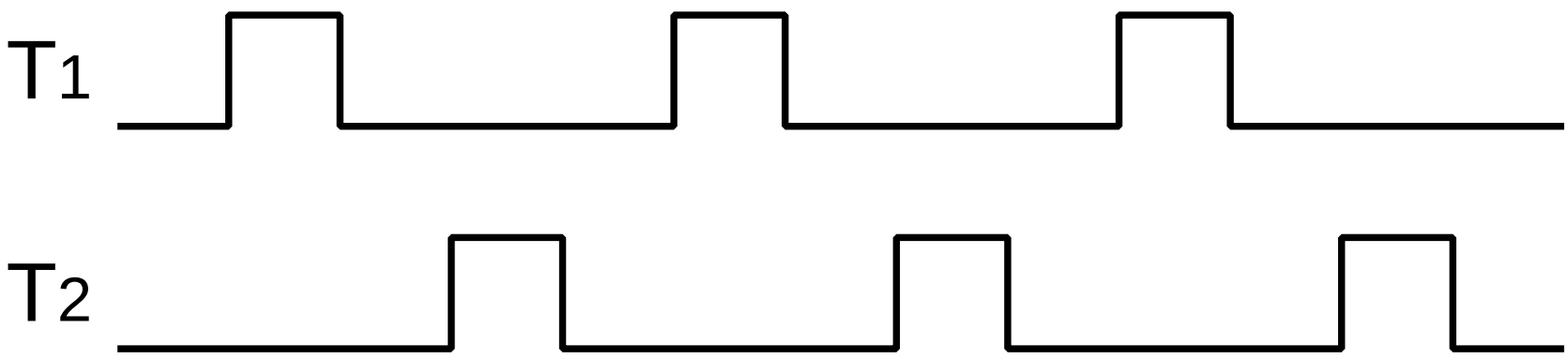

main memory
00000= 0
00001= 1
00010= 2
11111=31
Program Counter
Instruction Decoder
Address Decoder
3bit
5bit
add, subtract
Arithmetic Unit
8FA
load,
store
add, subtract
Instruction Register
stop,
jump,
jump
flag
Flag Register
Accumulator

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**

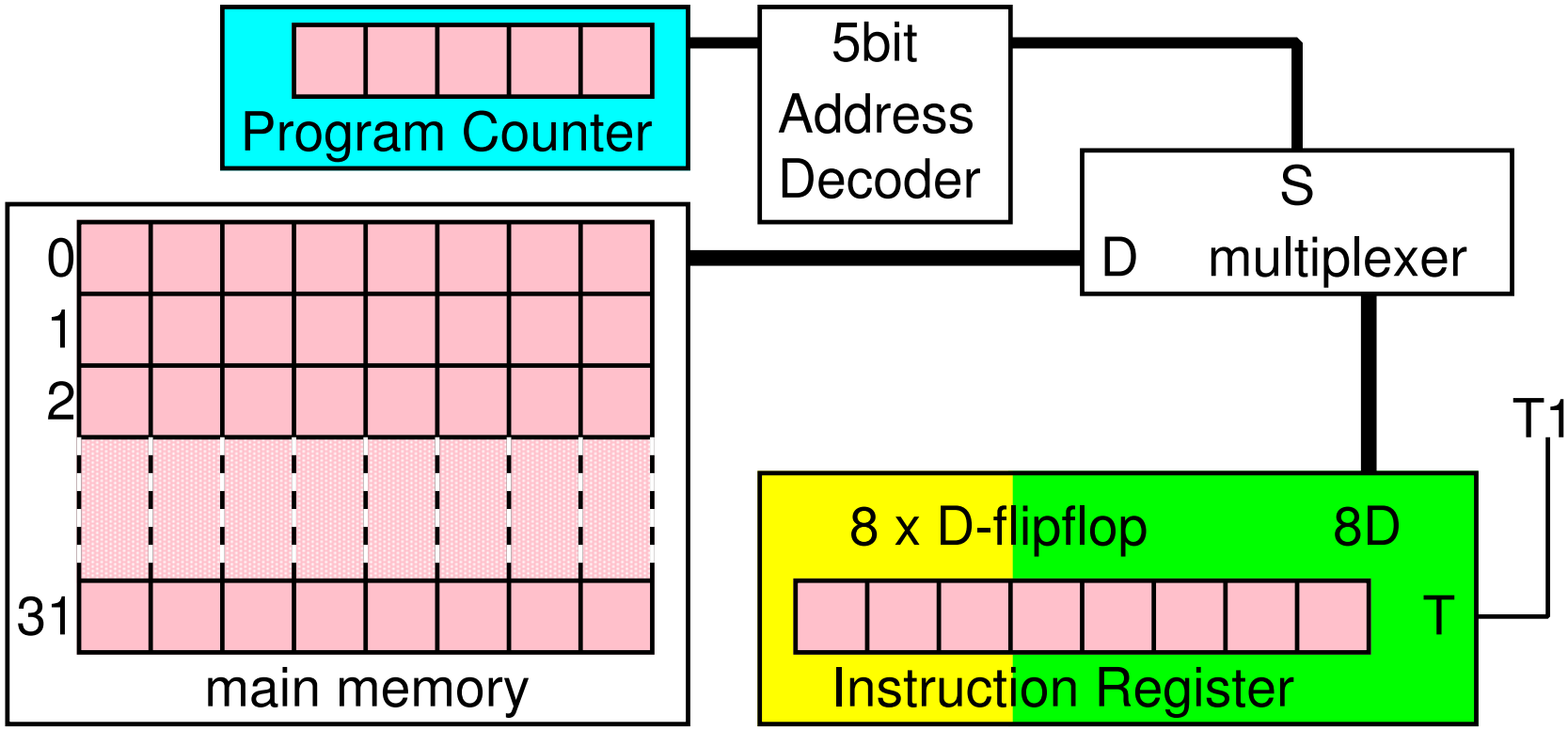
レジスタ・メモリの側から、

- **どの命令に対して**
- **何が書き込まれるべきか**

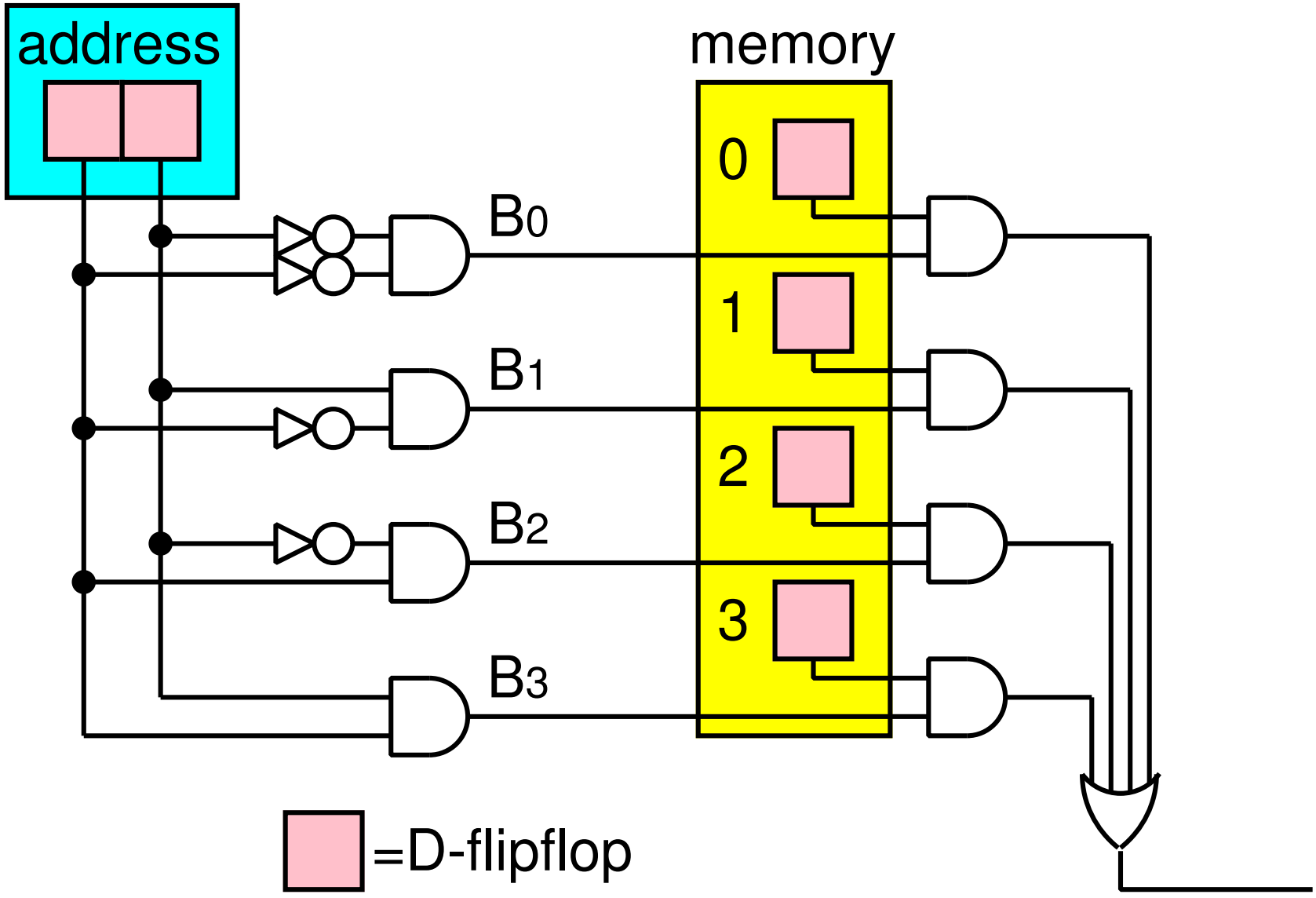
を見てみよう。

命令レジスタ (IR)

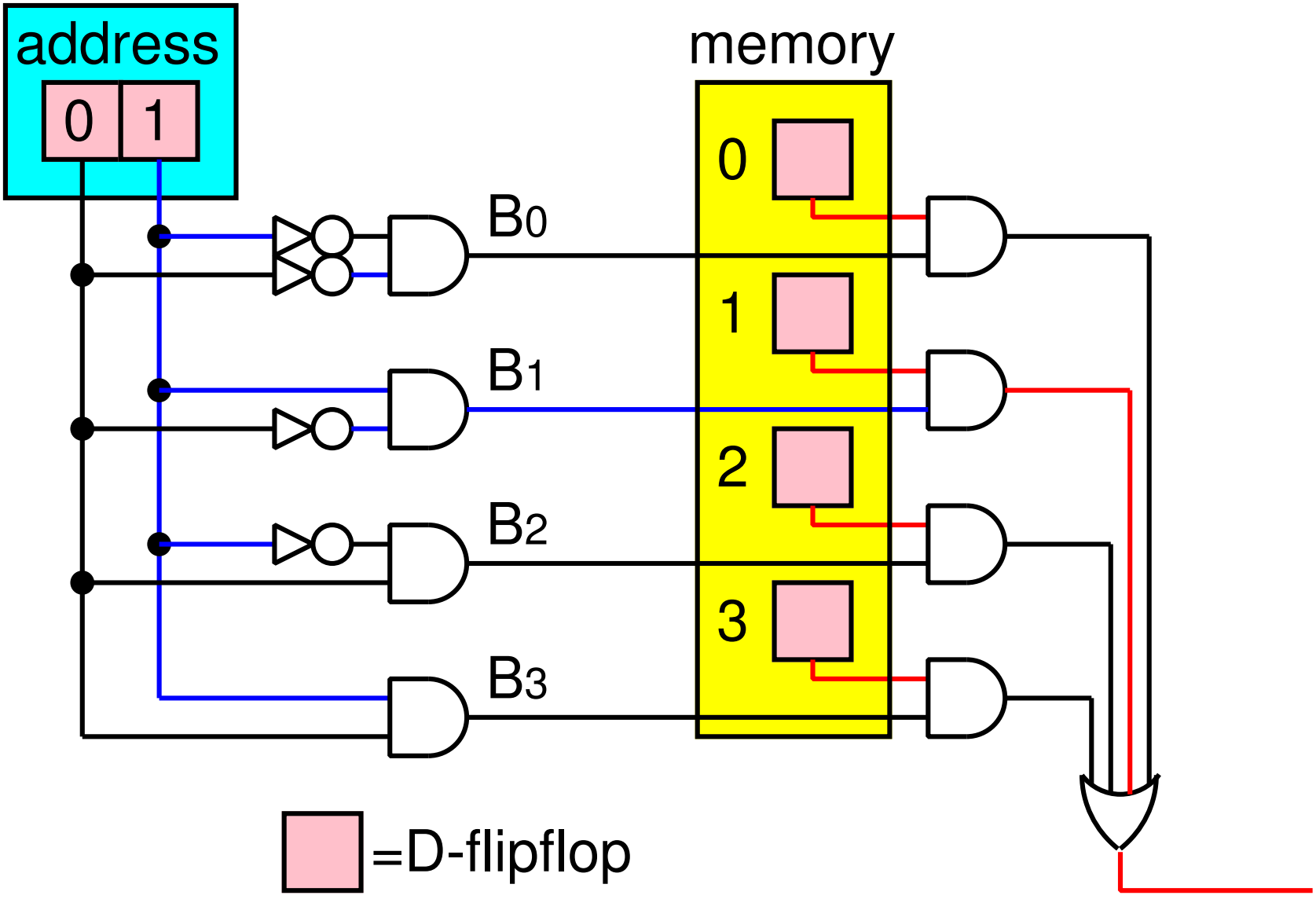
- パルス T_1:
  PC が指定した番地の主記憶の内容



データの読出し:



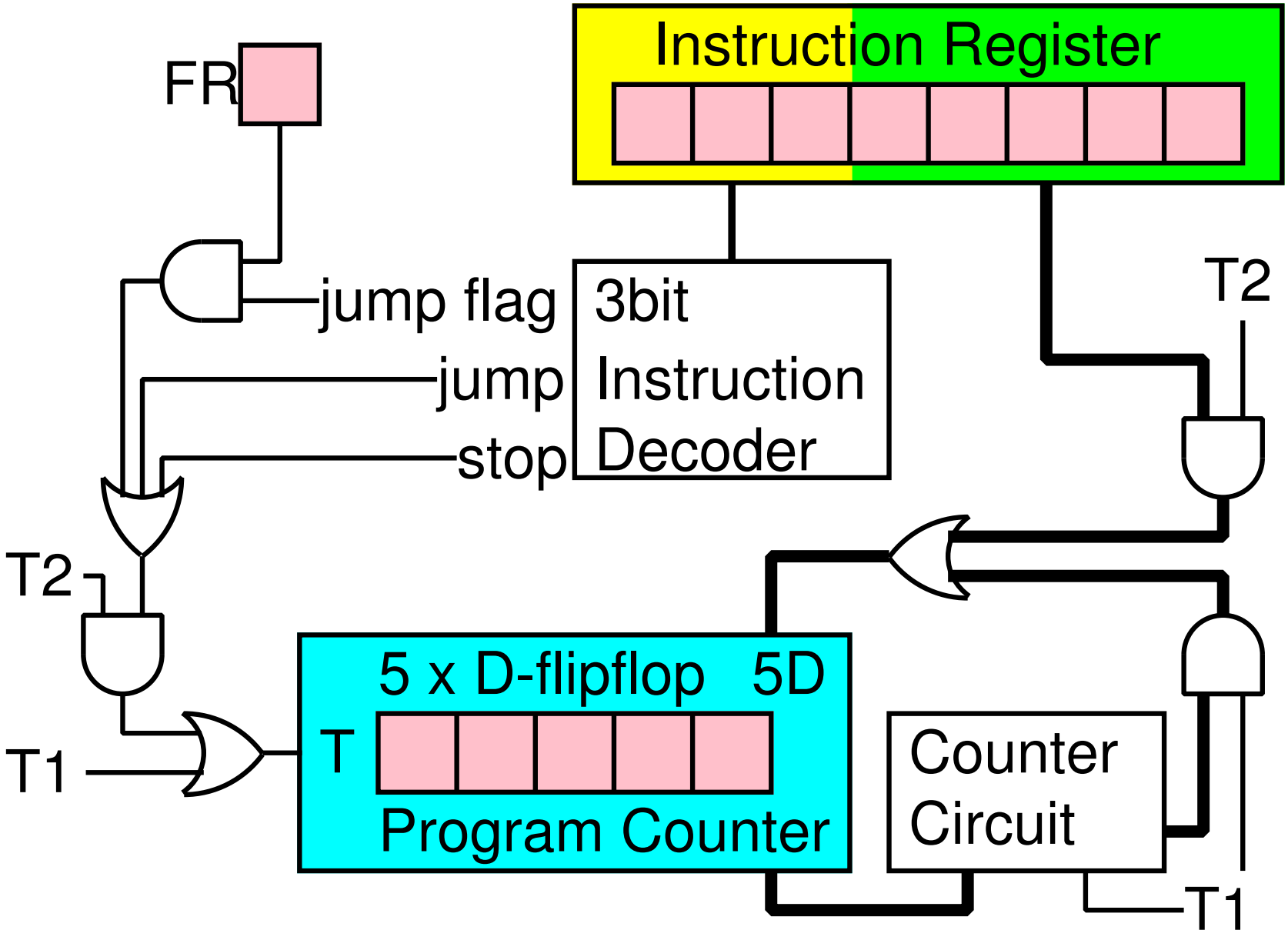
データの読出し: 1 番地のデータを読出し



プログラムカウンタ (PC)

- **パルス T_1:カウンタを進める**
  (カウンタ回路の出力)

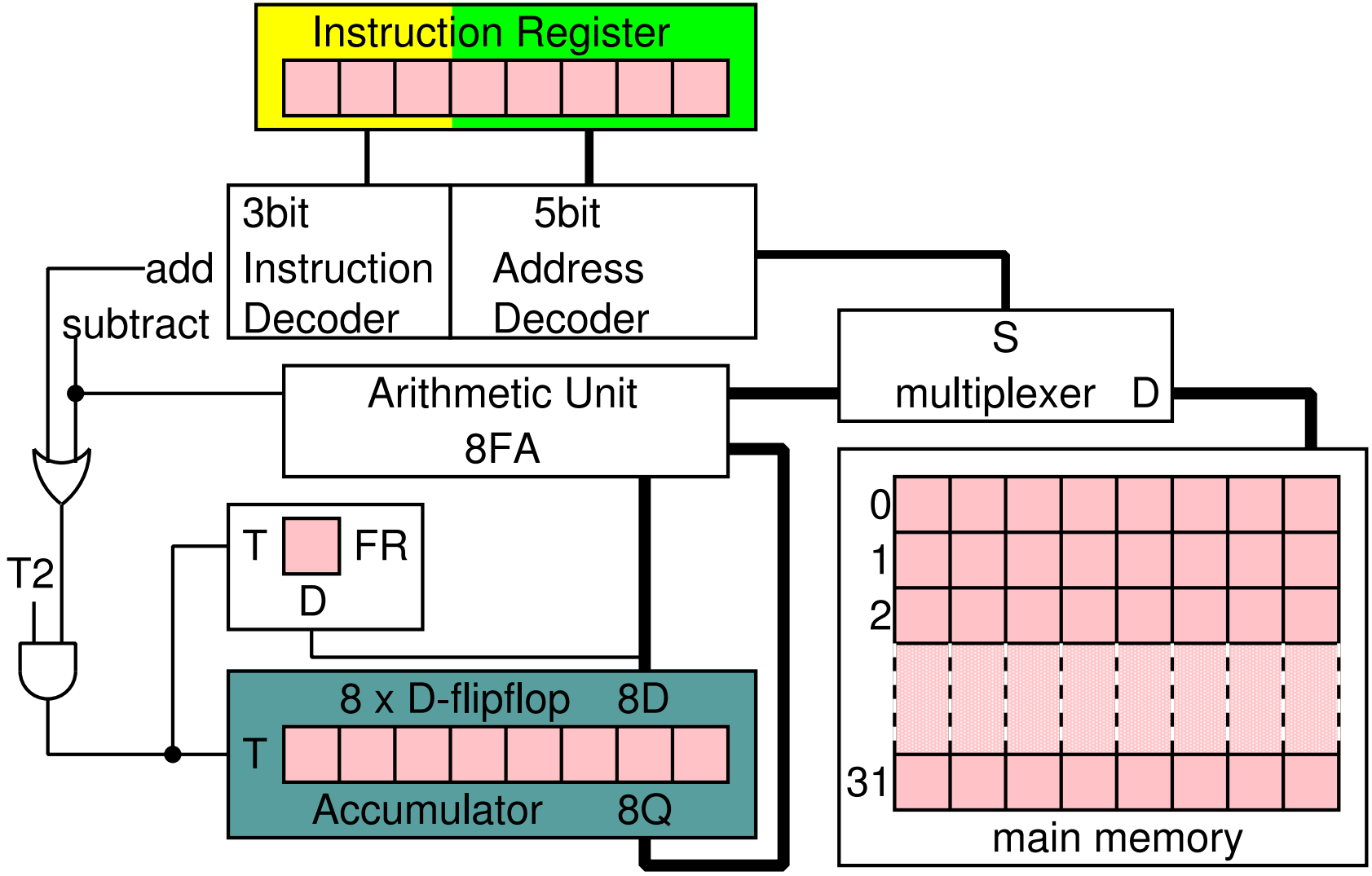
- `jump`: **命令レジスタの番地部**
- `jump flag`: **同上 (FR が 1 ならば)**
- `stop`: **同上**


Instruction Register
FR
jump flag
jump
stop
3bit
Instruction
Decoder
T2
T2
T1
5 x D-flipflop 5D
T
Program Counter
Counter
Circuit
T1

アキュムレータ (Acc)

- `load`:
  PC が指定した番地の主記憶の内容

- `add`・`subtract`:
  計算結果 (演算回路の出力)


Instruction Register
3bit
Instruction
Decoder
5bit
Address
Decoder
add
subtract
S
multiplexer D
Arithmetic Unit
8FA
T2
T
FR
D
8 x D-flipflop 8D
T
Accumulator 8Q
0
1
2
31
main memory

演算回路 (Arithmetic unit)

(レポート課題例の問 7 の拡張)

制御入力 `subtract` **が** 0 **か** 1 **かに従って、**
二進 8 **桁の加算または減算を行なう。**

$-B = \overline{B} + 1$ ($\overline{B}$ **は** B **の各 bit 反転**)

$\longrightarrow A - B = A + \overline{B} + 1$

$\longrightarrow$ **工夫すると加減算は回路を共通化できる**

計算結果の正負に応じて FR を 0 **／** 1 **に**

$\longrightarrow$ **符号 bit をそのまま出力**

演算回路 (Arithmetic unit)

(レポート課題例の問 7 の拡張)

制御入力 subtract **が** 0 **か** 1 **かに従って、**
二進 8 **桁の加算または減算を行なう。**

$-B = \overline{B} + 1$ (**$\overline{B}$ は B の各 bit 反転**)

$\longrightarrow A - B = A + \overline{B} + 1$

$\longrightarrow$ **工夫すると加減算は回路を共通化できる**

計算結果の正負に応じて FR を 0 **／** 1 **に**

$\longrightarrow$ **符号 bit をそのまま出力**

演算回路 (Arithmetic unit)

(レポート課題例の問 7 の拡張)

制御入力 subtract **が** 0 **か** 1 **かに従って、**
二進 8 **桁の加算または減算を行なう。**

$-B = \overline{B} + 1$ **(**$\overline{B}$ **は** B **の各 bit 反転)**

$\longrightarrow A - B = A + \overline{B} + 1$

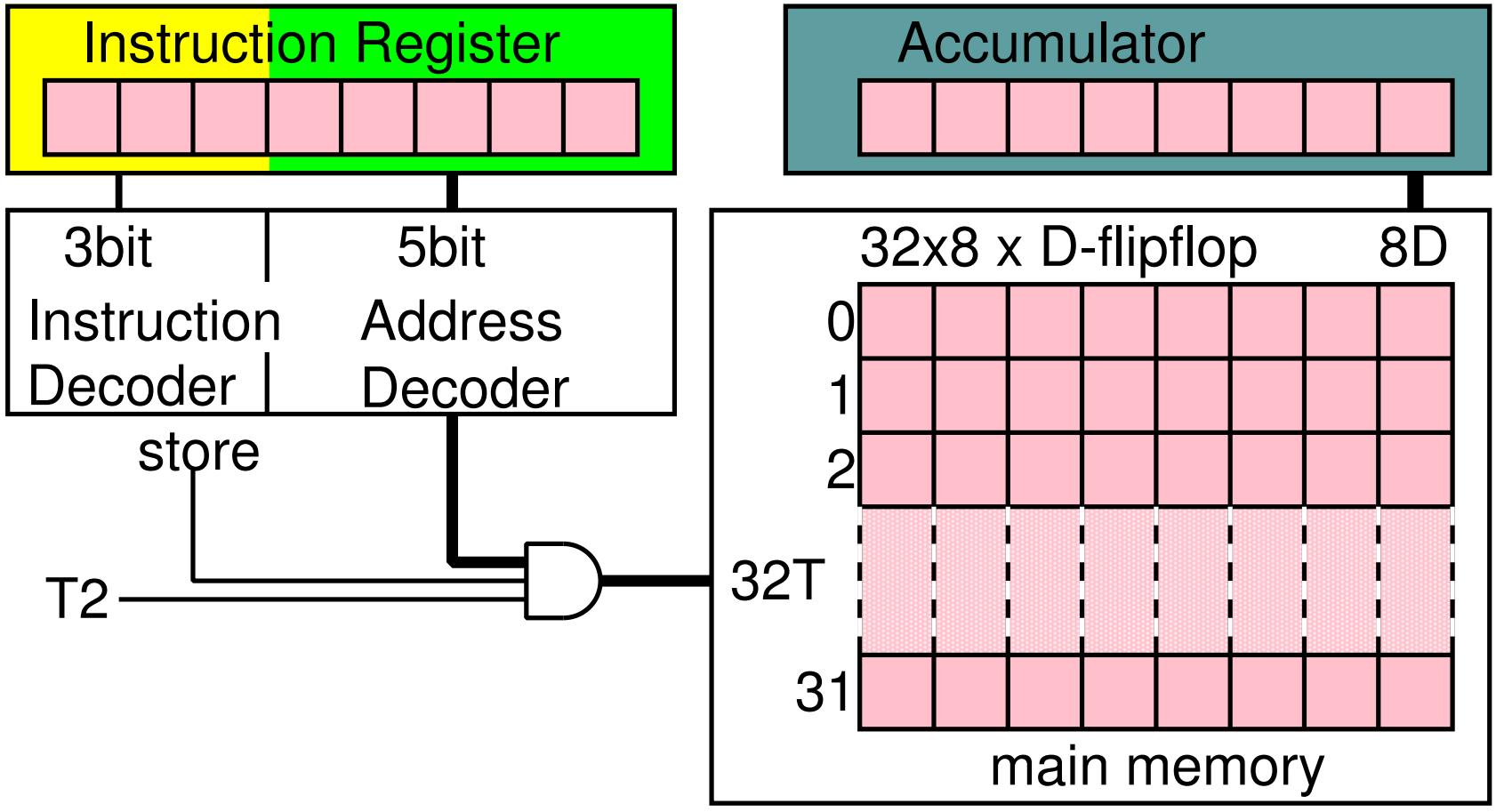
$\longrightarrow$ **工夫すると加減算は回路を共通化できる**

計算結果の正負に応じて FR を $0 \diagup 1$ **に**

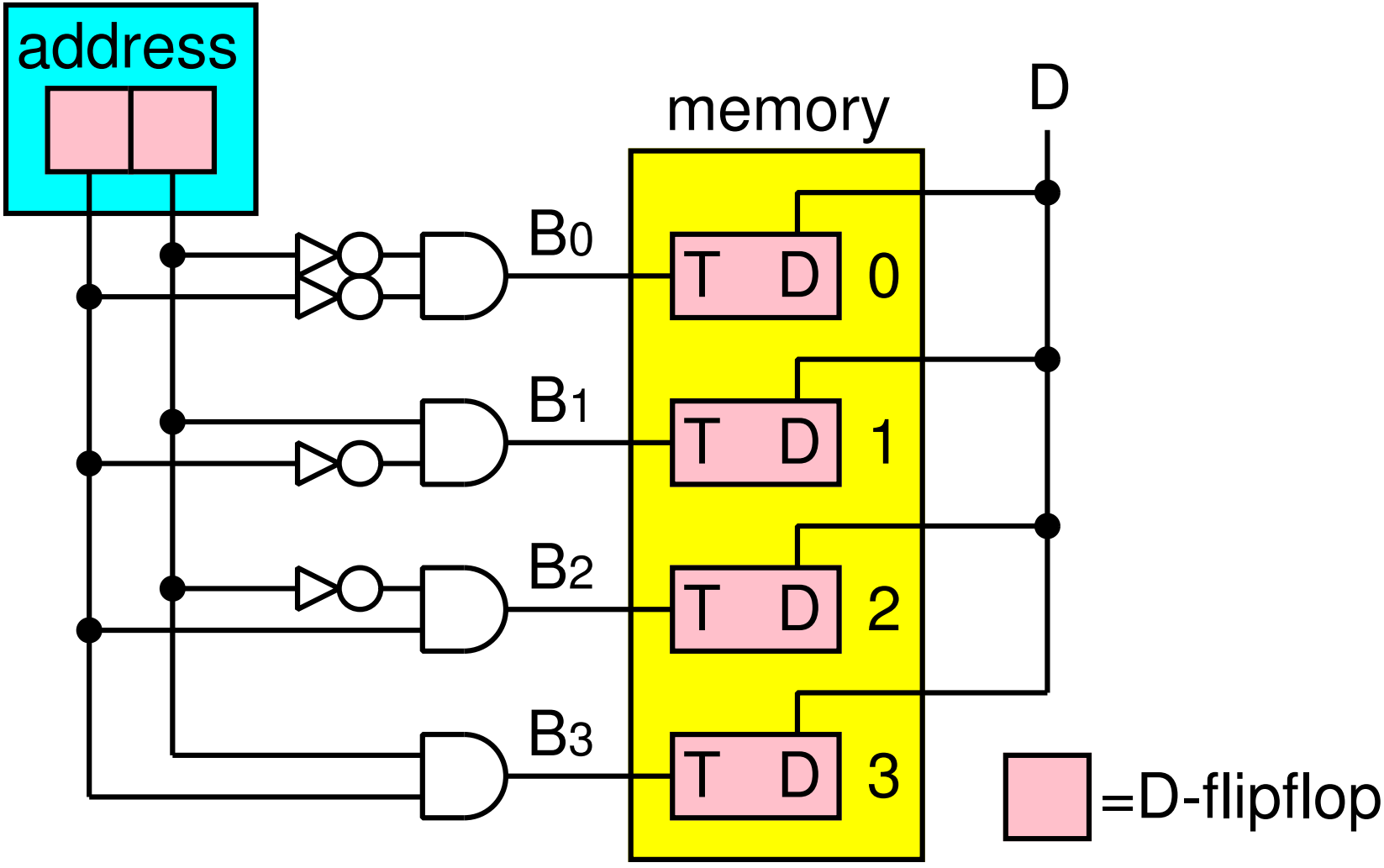
$\longrightarrow$ **符号 bit をそのまま出力**

主記憶 (main memory)

- `store`:
  Acc の内容を、IR が指定した番地に



データの書込み:



以上で、説明用の簡易モデルについて、

論理回路による実装が出来た!!

計算の実行:

予め

- 主記憶にプログラム・データを
- PC に実行開始番地 (通常 0 番地) を

それぞれ書き込んでおいて、

パルスを発生させると動作する。

プログラム・データ: bit 列 (機械語)

説明用の簡易モデルの機械語:

命令の符号化 (encoding)

命令	番号	符号化
stop	1	001
load	2	010
store	3	011
add	4	100
subtract	5	101
jump	6	110
jump flag	7	111

例: $87 + 26$ を計算する

	番地	機械語 二進	機械語 十六進
0	00000	01000100	0x44
1	00001	10000101	0x85
2	00010	01100110	0x66
3	00011	00100000	0x20
4	00100	01010111	0x57
5	00101	00011010	0x1a
6	00110	00000000	0x00

例: $87 + 26$ を計算する

	機械語		アセンブリ言語	
	番地	二進	十六進	
0	00000	01000100	0x44	load A
1	00001	10000101	0x85	add B
2	00010	01100110	0x66	store C
3	00011	00100000	0x20	stop 0
4	00100	01010111	0x57	A 87
5	00101	00011010	0x1a	B 26
6	00110	00000000	0x00	C 0

例: $87 + 26$ を計算する

	番地	機械語 二進	機械語 十六進	アセンブリ言語
0	00000	01000100	0x44	load A
1	00001	10000101	0x85	add B
2	00010	01100110	0x66	store C
3	00011	00100000	0x20	stop 0
4	00100	01010111	0x57	A 87
5	00101	00011010	0x1a	B 26
6	00110	00000000	0x00	C 0

アセンブリ言語 (assembly language):
機械語と一対一対応する符丁 (ニーモニック) で
プログラムを記述したもの

アセンブラ (assembler):
アセンブリ言語で記述されたプログラムを
機械語に直すプログラム

$\longrightarrow$ **CPU のアーキテクチャによって**
それぞれ異なる (互換性がない)