計測工学　　2010年後期

デジタル信号処理と計測工学

By　小林春夫

群馬大学大学院工学研究科

電気電子工学専攻

電話　0277 30 1788

部屋　電気電子棟　402号室

e-mail: k_haruo@el.gunma-u.ac.jp

トランジスタの発明者
ウィリアム・ショックレー

米国の物理学者。
1910年2月13日、
英国生まれ。
「トランジスタの父」と呼ばれて
1989年8月12日没。

シリコン・バレーの発祥

- ウィリアム・ショックレーはスタンフォード大学卒業後、AT&Tのベル研究所に入る。
- 1947年に研究員の仲間と接合型トランジスタを発明。
- この業績が評価され、

ショックレー、ジョン・バーディーン、ウォルター・ブラッテンに、1956年にノーベル物理学賞が授与される。

- 1953年、ベル研究所を去り、

1955年にはサンフランシスコ郊外に

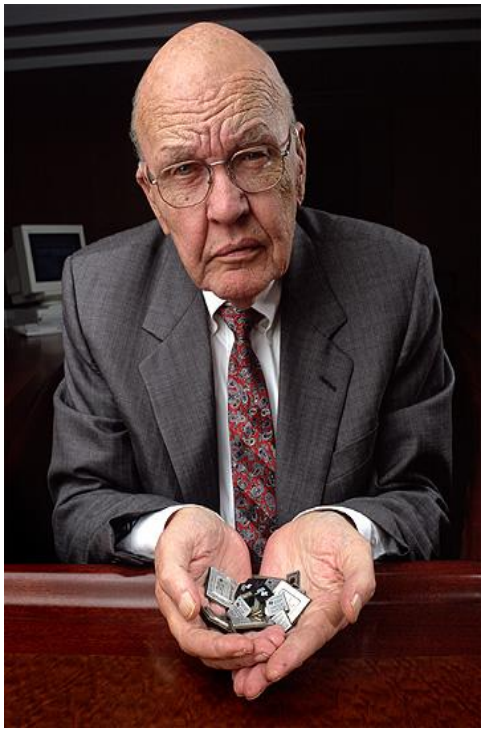
ショックレー半導体研究所を設立。

同研究所の周辺に半導体産業が集まりはじめ、

シリコンバレーと呼ばれる半導体産業のメッカを形成。

ICの発明者　ジャック・キルビー

Texas Instruments(テキサス・インスツルメンツ)社
のJack Kilby(ジャック・キルビー)氏が
半導体集積回路を発明。

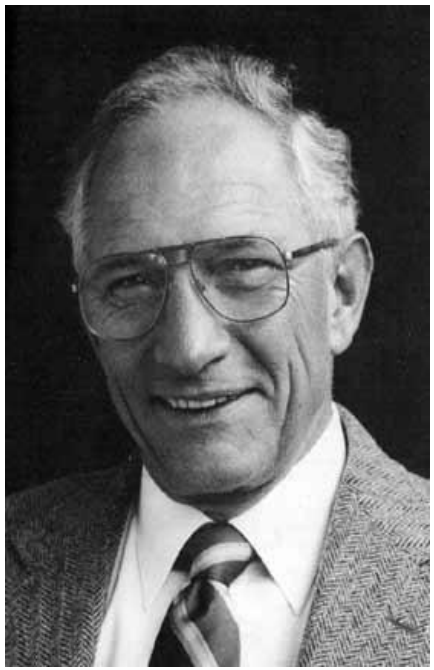


1958年の発明、
アメリカでは1959年に出願、1964年に登録。
日本では1960年に出願、1965年に公告。

この業績により2000年にノーベル賞を受賞。

もう一人のICの発明者
ロバート・ノイス

米国の半導体技術者。
1927年12月12日、
アイオワ州バーリントン生まれ。
半導体集積回路の発明者の
一人として、
Intel社の共同創業者とて
知られている。
1990年6月3日没。

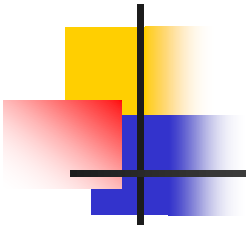


ロバート・ノイス

マサチューセッツ工科大学を卒業したノイスは、
ウィリアム・ショックレー博士の直接の誘いを受けて、
1956年からショックレー研究所に勤めた。
しかしショックレーとの方針の違いが顕著になると、
ゴードン・ムーアらと共に同研究所を去り、
1957年、新たにFairchild Semiconductor社を創立。
Fairchild社で半導体メモリーの研究開発と普及に努める。
ほどなくして出資親会社と意見が衝突。
ノイスはムーアと共にFairchild社を去り、Intel社を創設。

インテル社とロバート・ノイス

Intel社で半導体メモリーを中心に集積回路の
研究開発を続けた。
トランジスタの表面を酸化シリコンの皮膜で覆う
プレーナー法を開発し、特許を取得。
マイクロプロセッサの研究開発も進められ、
1970年にはIntelが世界初のDRAMを販売するなど、
世界一の半導体企業の名声を揺るぎないものにした。
ノイスは1970年まで社長・会長職に就き、
「シリコンバレーの主」と称された。



DSPとは何か

Digital Signal Processor

デジタル信号処理チップ

Digital Signal Processing

デジタル信号処理

自然界の信号は全てアナログ

ex. 音声、電波、電圧、電流、

デジタル信号処理システム

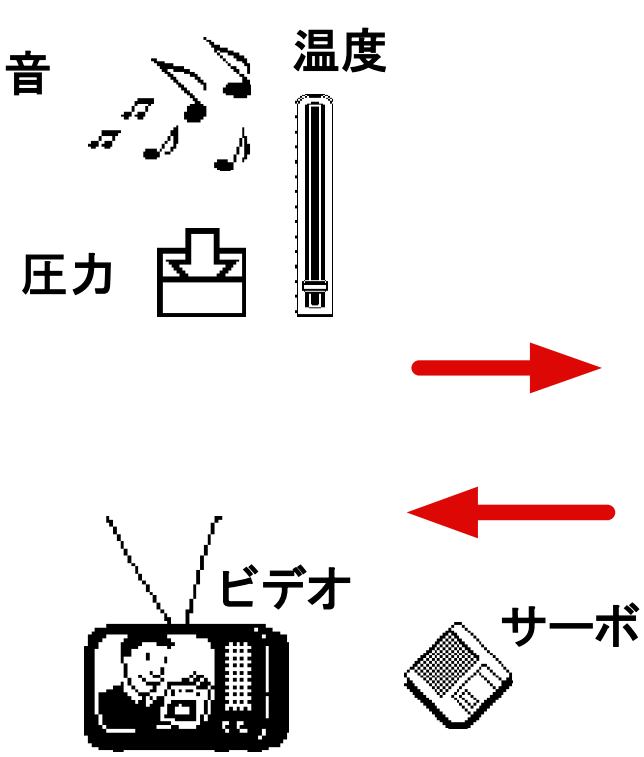
AD変換器： アナログ･デジタル変換器
(Analog-to-Digital Converter: ADC)

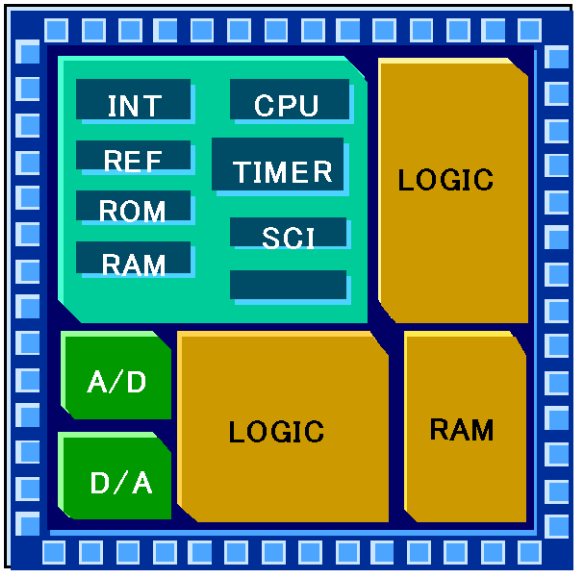
DA変換器： デジタル･アナログ変換器
(Digital-to-Analog Converter: DAC)

自然界の信号はアナログ

自然界の信号は
アナログ

LSIでの信号処理は
デジタル





例： 音声信号をなぜデジタル処理するのか

デジタル処理の長所

田中紘資先生
作成資料

高機能の実現

多様性　→　任意の計算処理が可能で複雑な処理が容易。
融通性　→　適応処理や時間処理など、処理形態が豊富。
発展性　→　誤り訂正付加や暗号化など、処理形態が豊富。

高性能の実現

高精度　→　高Ｓ／Ｎが容易で、高品質な記録・再生が容易。
安定性　→　温度・経時変化による劣化が無く、保守が容易。
小型化　→　高集積ＬＳＩ化容易で、システムの小型化が可能。

高生産性の実現

設計容易性　→　ＣＡＤ設計自動化による開発効率向上が容易。
製造容易性　→　ばらつきが少なく、無調整化が可能。

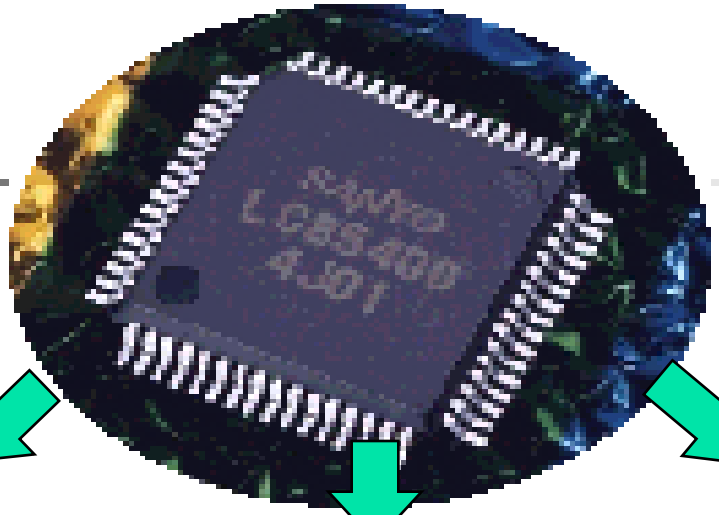
音声録音再生LSI応用商品

コードレス留守番電話

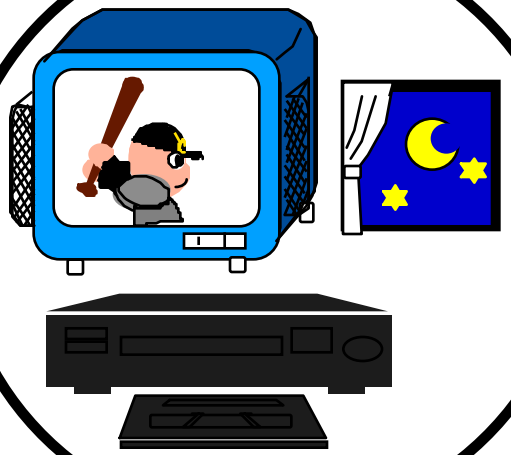
―― 特長 ――

- ＤＳＰデジタル録音方式（用件応答メッセージ録音）
- 遅聞き・早聞き再生機能
- 通話録音機能
- ひとこと伝言機能
- 固定応答メッセージ
- 操作ガイダンス

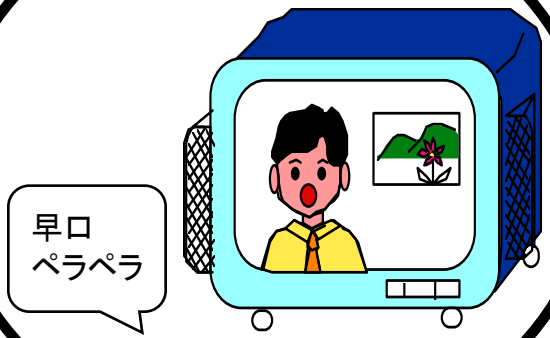
話速変換LSIの事例



「短時間」で見れる



「ゆっくり」聞ける



Hello Do you understand ?

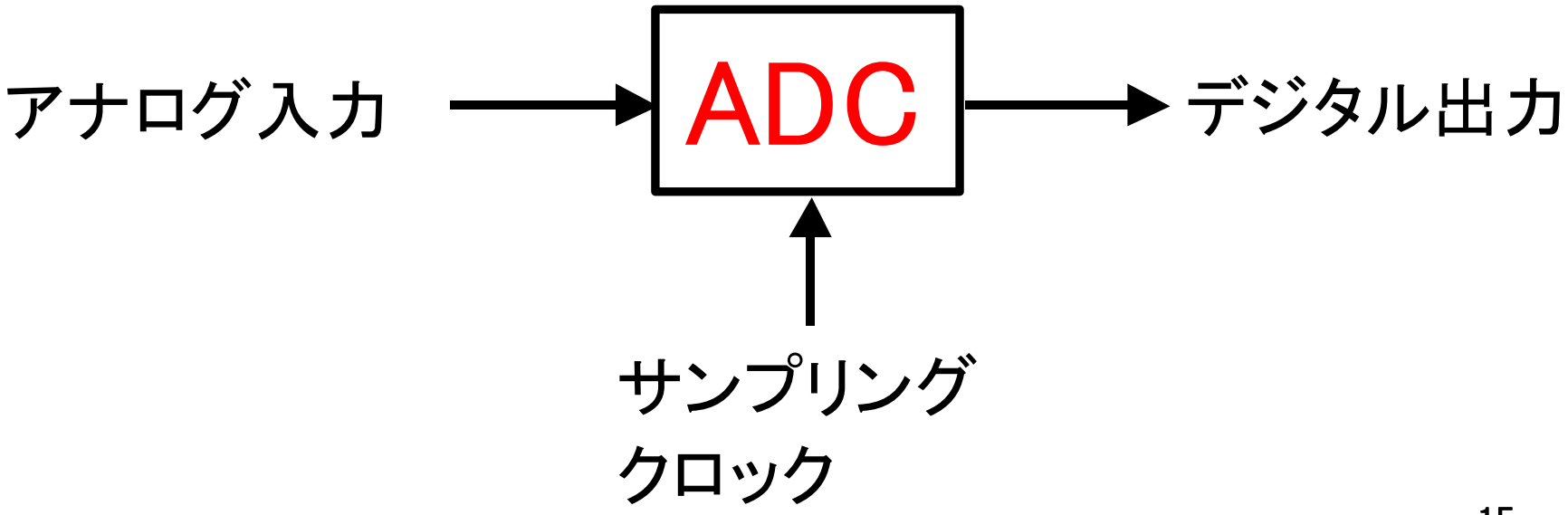
短時間で聞ける

デジタル信号処理
Digital Signal Processing

DSPとは
デジタル表現された信号とその処理方法に関する研究分野。
音響信号処理、画像処理、音声処理の三つの領域。
目標は実世界の連続的なアナログ信号を計測し、選別すること。
第一段階でアナログ-デジタル変換回路を使って信号を
アナログからデジタルに変換。
最終的な出力は別のアナログ信号であることが多く、
そこではデジタル-アナログ変換回路が使用。
DSPで実行するアルゴリズムは専用のコンピュータを使うことが多い。
デジタルシグナルプロセッサという特殊なマイクロプロセッサが
使われ、こちらも**DSP**と略記される。
DSP向けに最適化されており、リアルタイムで信号を処理する。

AD変換器の動作

アナログ信号（電波、音声、電圧、電流等を
デジタル信号（0, 1, 1, 0, ...）に変換する。



アナログ信号とデジタル信号

アナログ信号

連続的な信号

例：　自然界の信号（音声、電波）、

アナログ時計　（直観的にすぐ時間がわかる）

「坂道」

デジタル信号

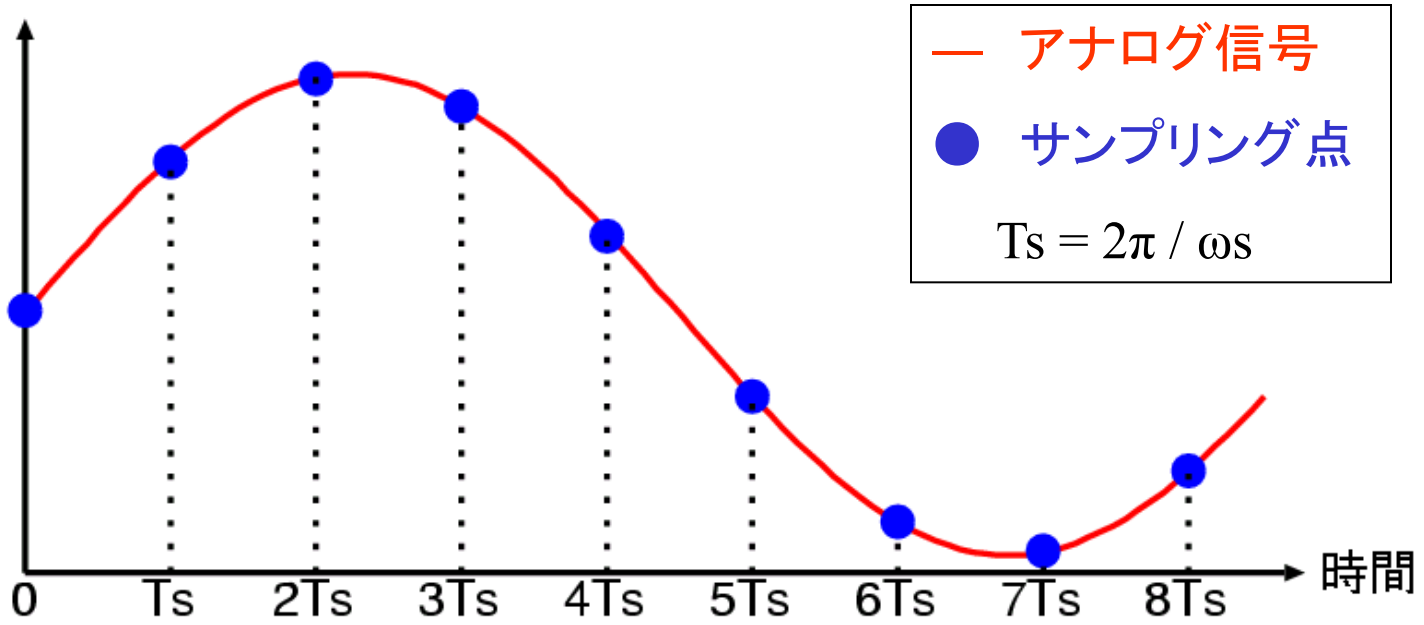
離散的・数値で表現された信号

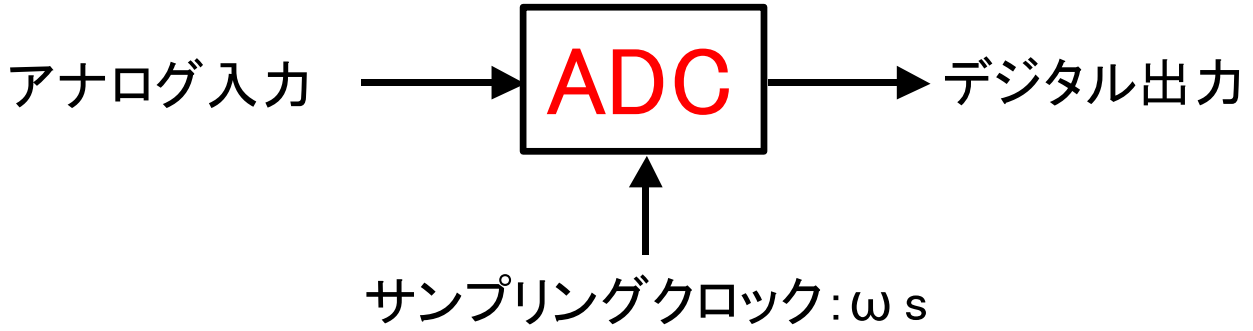
例：コンピュータ内での２進数で表現された信号

デジタル時計　（精度がよい）

「階段」

時間の量子化（サンプリング）





サンプリング定理

アナログ信号波形Ｘ（ｔ）が、０～Ｗ［Ｈｚ］の間に帯域制限されているとき、
Ｘ（ｔ）をＴ＝１／２Ｗ［Ｓｅｃ］ごとに標本化すれば、標本値系列から
次式のように、元の波形が完全に再現できる。

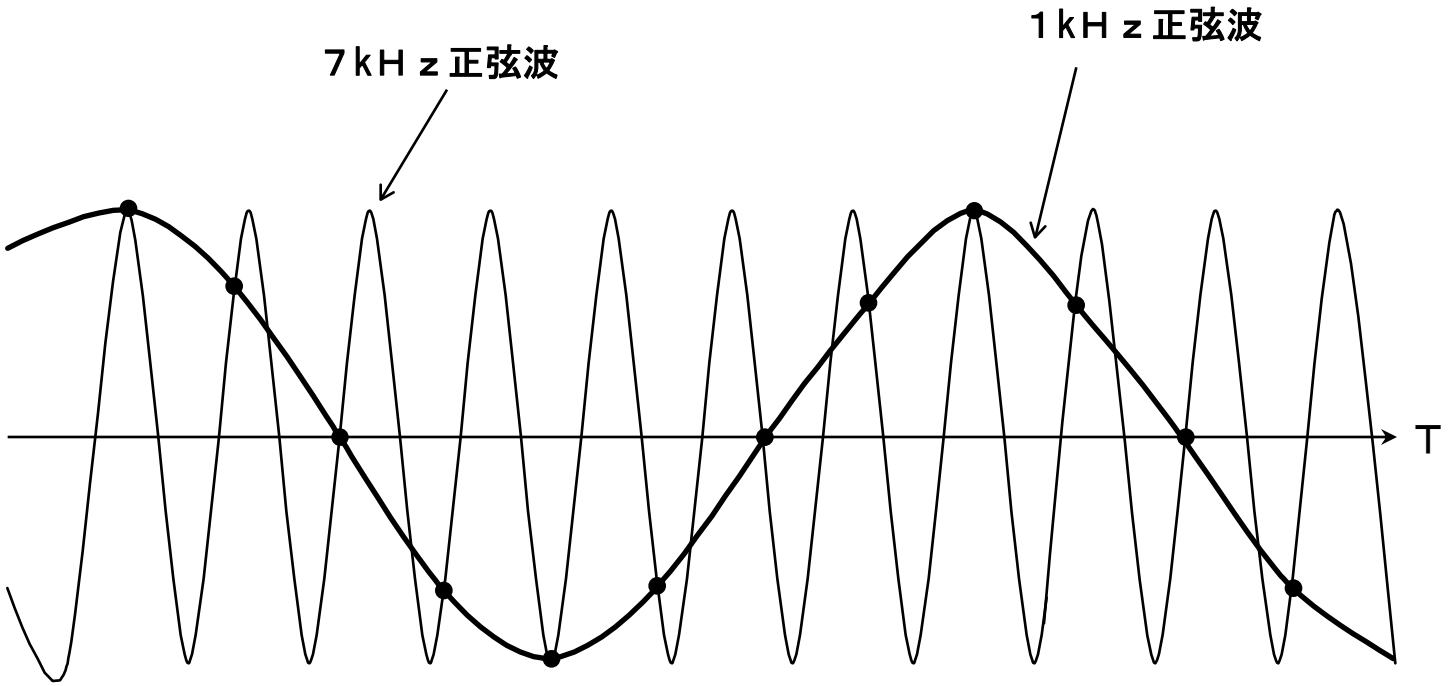
$$X(t) = \sum_{n=-\infty}^{\infty} X(n/2W) \cdot \frac{Sin\{2\pi W(t - n/2W)\}}{2\pi W(t - n/2W)}$$

Ｔ＝１／２Ｗ　：　標本化周期

Ｘｎ＝Ｘ（ｎＴ）　：　標本値

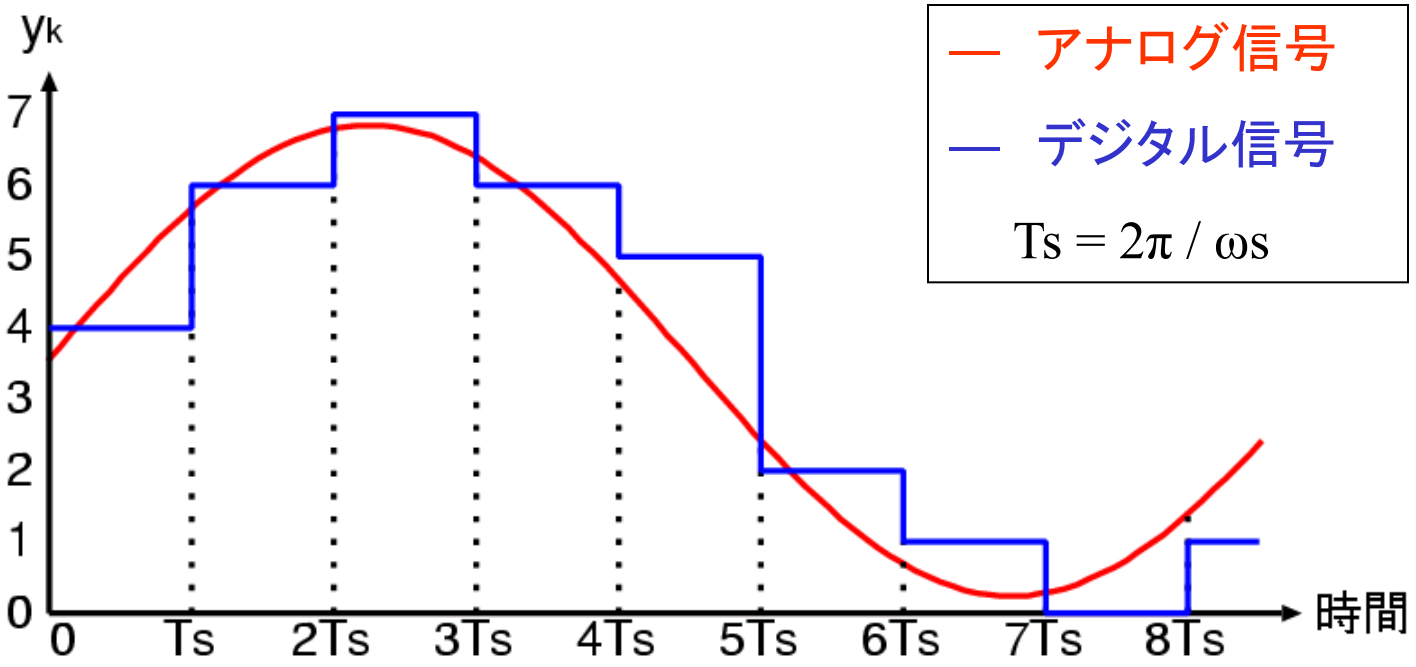
サンプリングと折り返し（aliasing）

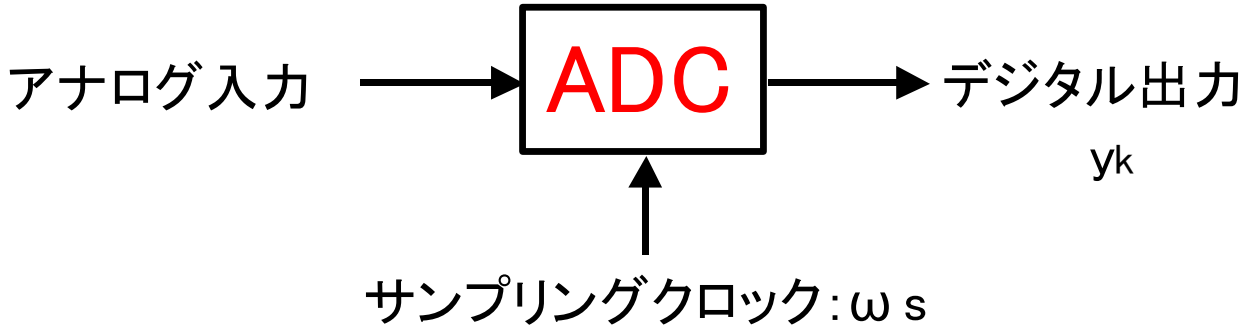
- ８ｋＨｚサンプリングを行うと、１ｋＨｚと７ｋＨｚは区別できない。



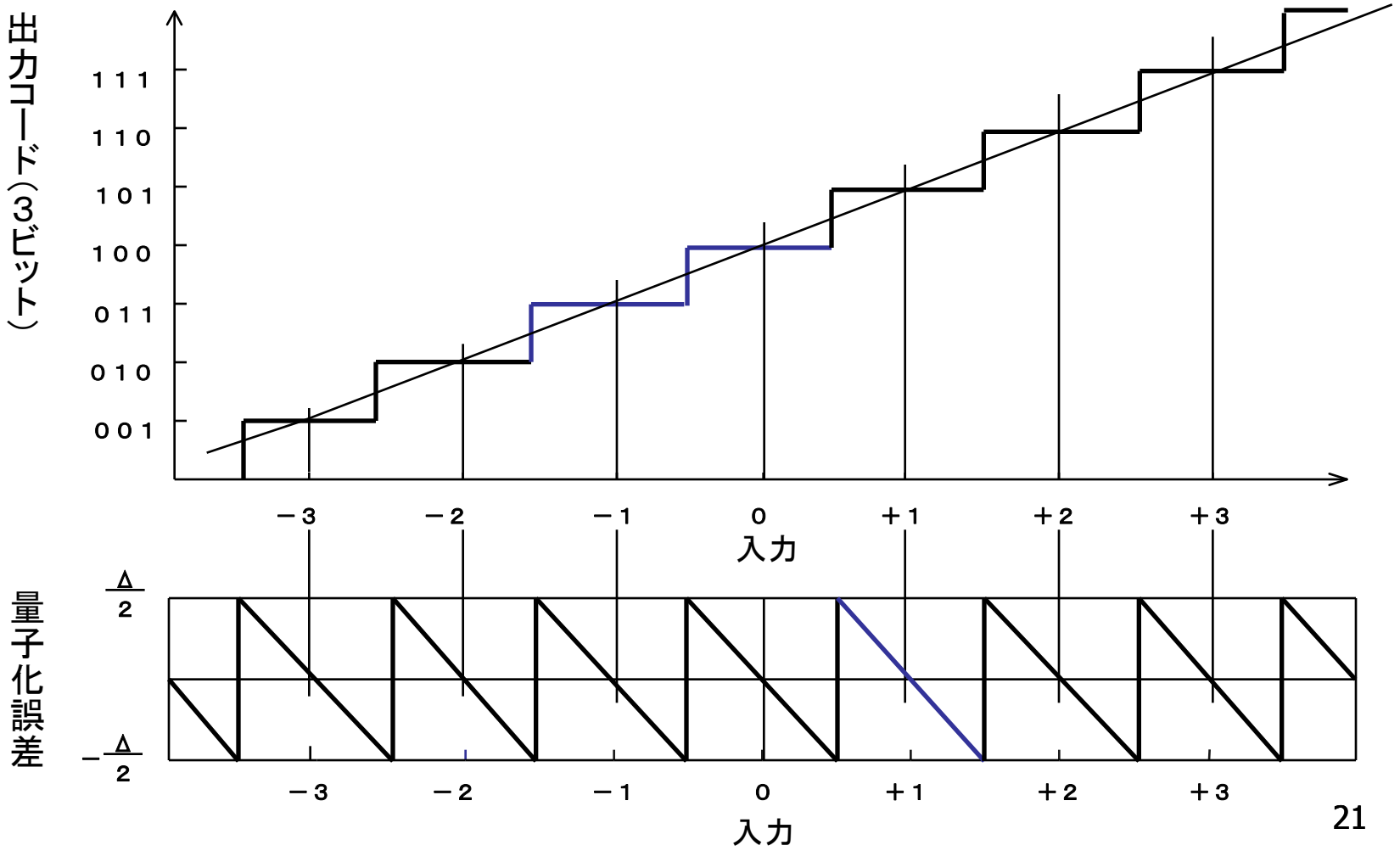
- は８ｋＨｚサンプリング値を表す。

空間の量子化
（信号レベルの数値化）

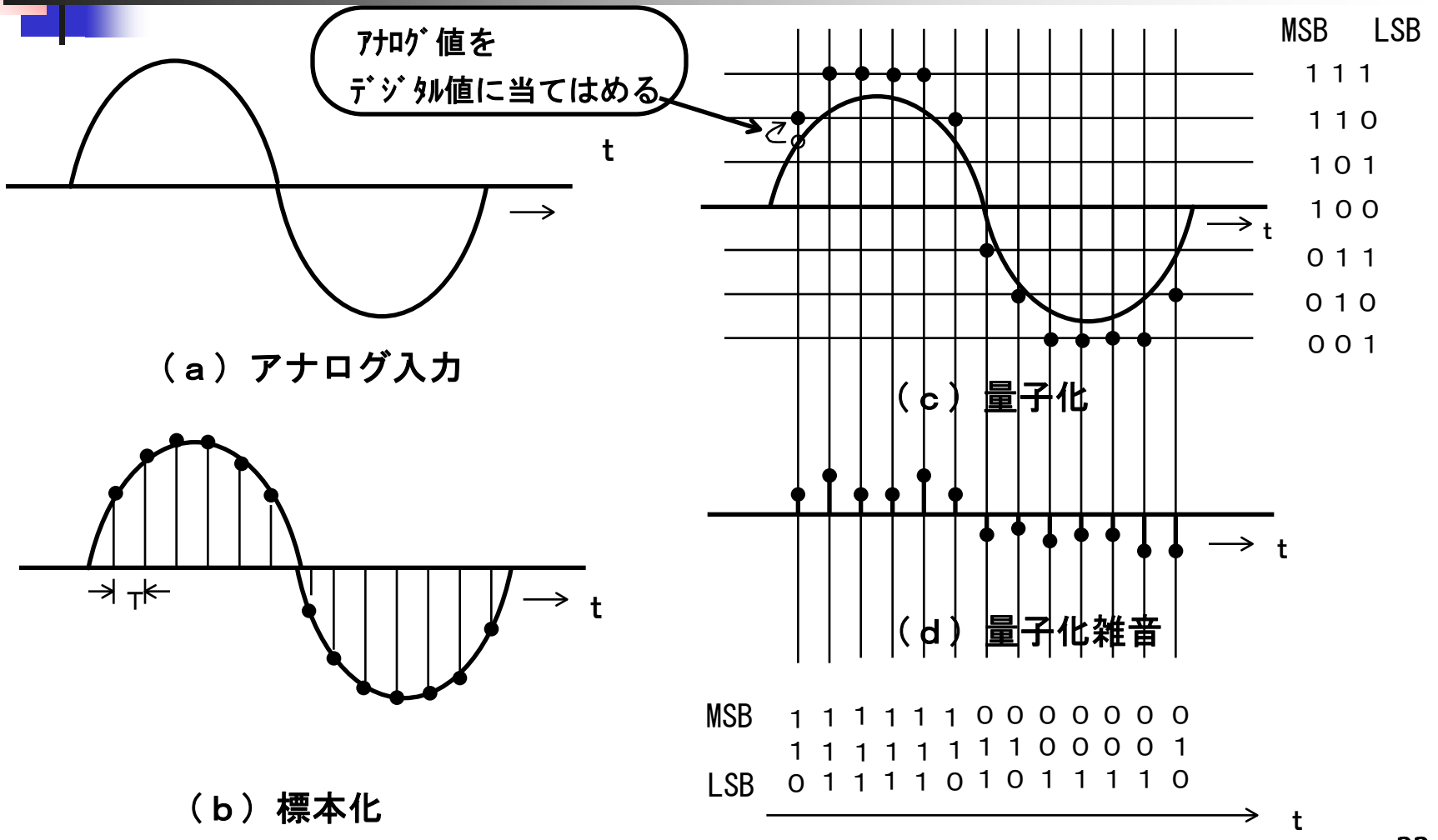




理想ＡＤ変換器の量子化誤差

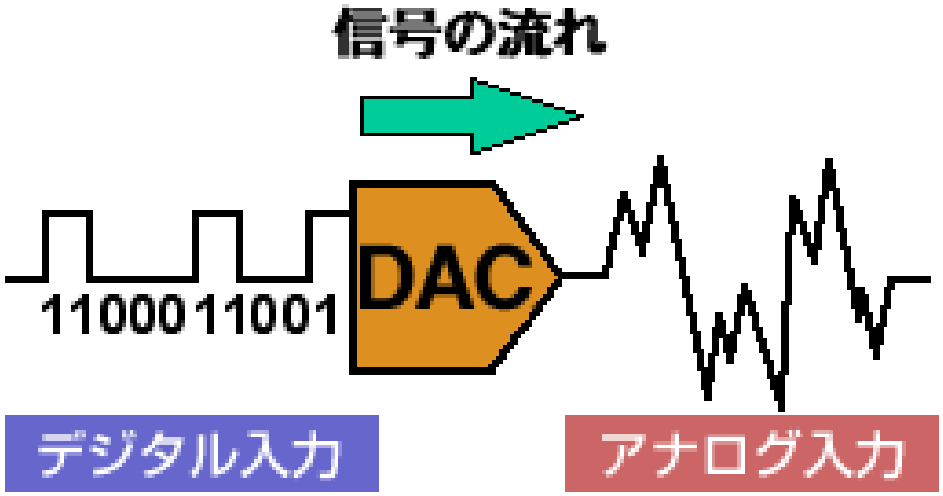


アナログ　-> デジタル　変換波形

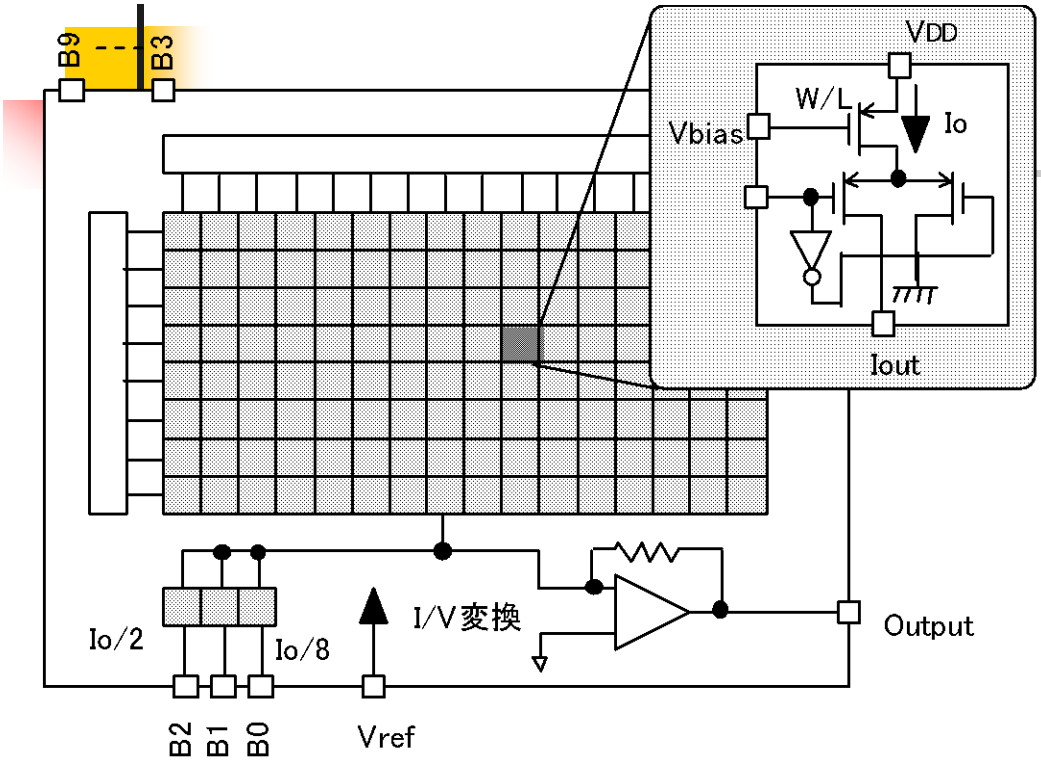


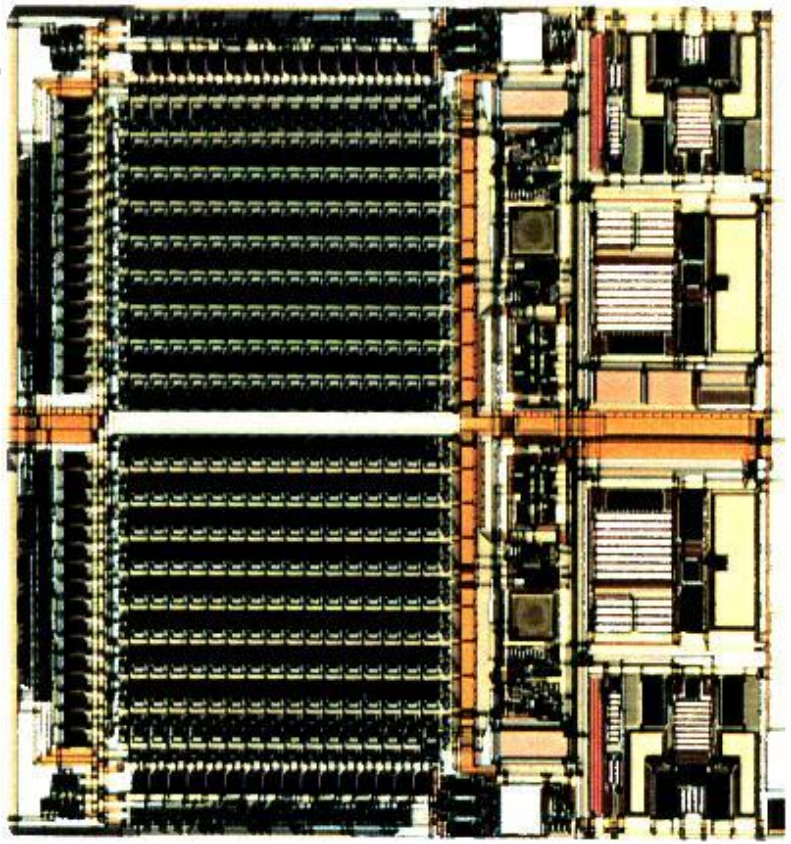
DA変換器 (Digital to Analog Converter)

離散的なデジタル値を連続的なアナログ信号に変換する回路



サーボ用10ビット電流型DA変換器





DA変換器の性能

分解能	10 bit
積分直線性誤差	+0.11/−0.07 LSB
微分直線性誤差	+0.35/−0.47 LSB
	0.9 us (@5V)
消費電力	9.7 mW (@3V)
整定時間	21 mW (@5V)

0.8 um CMOS　1.31 mm^2

アナログ信号処理と デジタル信号処理

「アナログ信号処理は　無限の精度がでる」
というのは大きな誤り。

アナログ信号処理は
素子のノイズ、非線形性等のため精度はでない。

アナログ信号処理がデジタル信号処理と競合して負けるのは精度がでないことが大きな理由。

実務経験を積めばすぐわかる。

アナログ信号処理は（デジタルではまだできない）
高速・高周波信号処理の部分等に用いられる。

DSPチップの特徴（1）

デジタル信号処理アルゴリズム

例： FFT, デジタル・フィルタ

積和演算　x0・h0+x1・h1+x2・h2+ …+xn・hn

DSPチップ：　積和演算が得意

（はさみ）　（紙をきる）

マイクロ・プロセッサ：汎用的なデジタル処理

（包丁）

DSPチップの特徴(2)

- デジタル乗算器(掛け算器)内蔵

  積和演算　x0･h0+x1･h1+x2･h2+ …+xn･hn

  の積を高速に実行。

  High-end のDSP チップは複数の掛け算器をもつ

- ハーバード･アーキテクチャ

フォン･ノイマンのボトルネックを解消。

- 並列処理　(Parallel Processing)

  → 皆で一緒(同時)に仕事をすれば　早く済む。

デジタル乗算

２進数の乗算

```
   0101   (5)
x) 1011   (11)
-------------
   0101
  0101
 0000
0101
-------------
0110111    (55)
```

加算器だけで
乗算を行うと
何サイクルも要する。

乗算器なら
１サイクルでできる。

デジタル加算器の実現(1)
（半加算器； Half Adder）

2入力2進加算

A	0	1	0	1
+) B	+) 0	+) 0	+) 1	+) 1
Co S	0 0	0 1	0 1	1 0

真理値表

A	B	Co	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

1ビット加算器の実現

$S = A \oplus B$

$Co = A \cdot B$

デジタル加算器の実現(2)
（全加算器； Full Adder）

3入力2進加算

A　入力1

B　入力2

+) Cin　下からの繰り上がり

Co S

$S = A \oplus B \oplus Cin$

$Co = B \cdot Cin + A \cdot Cin + A \cdot B$

(Co は A, B, Cin の多数決）

真理値表

A	B	Cin	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
1	0	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

全加算器 (Full Adder)の補足説明

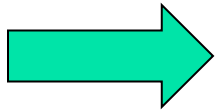
Cin: Carry in (下位の桁からの繰り上げ)

S: Sum（加算結果）

Cout: Carry out （上位の桁への繰り上げ）

```
   0   1
+) 1   1
-----------
 1 0   0
```

を考える。



全加算器の演算

```
   1
   0   1
+) 1   1
-----------
 1 0   0
```

全加算器の補足説明(続き)

```
    0  1  0  1  (5)
+)  0  1  1  1  (7)
-------------------
 0  1  1  0  0  (12)
```

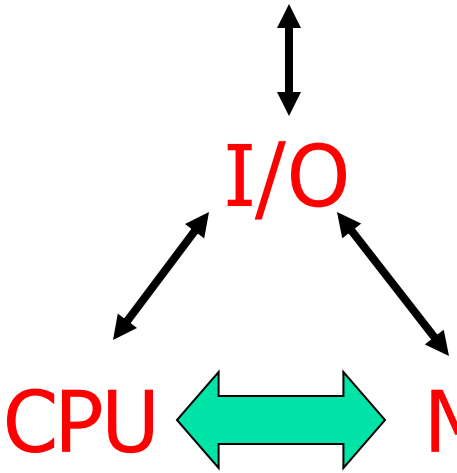
を考える。

```
    1  1  1
    0  1  0  1
+)  0  1  1  1
-------------------
 0  1  1  0  0
```

Carry（桁上げ）

Sum（加算結果）

デジタル・コンピュータ
ノイマン型アーキテクチャ



ノイマンのボトルネック

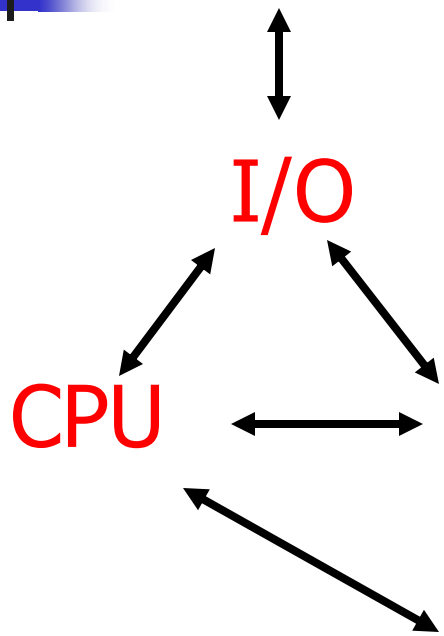
I/O: Input/Output
外部とのデータの入出力

CPU: 演算

Memory: データ、
プログラムの格納

- 大部分のデジタル・コンピュータの構成

デジタル・コンピュータ ハーバード型アーキテクチャ

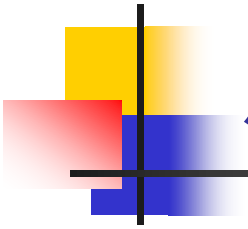


ハードウェア複雑

データ格納(h0,x0,h1,x1,....)

プログラム格納(式)

- ノイマンのボトルネック解消



ハーバード・アーキテクチャ

- 命令（プログラム）用とデータ用に物理的に分割されたメモリ（記憶装置）と信号通路を用いる。
- DSPに加えて、汎用マイクロコントローラの多くもハーバード・アーキテクチャをベース。
- 最新のマイクロプロセッサも

ハーバードとフォンノイマン両方のアーキテクチャを取り入れている。

2進数とデジタル

デジタルコンピュータは

　なぜ2進数を用いるのか　？

2つの状態は電子回路で実現しやすい。

例：　電圧の“高い”と“低い”

　　電流の“流れている”と“流れていない”

　　パルスの“ある”と“なし”

一方を“1”　他方を“0”と割り当てる

１０進数と２進数

10進	2進	10進	2進
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

例　2進数　1011　を
　１０進数に変換
1x2x2x2+0x2x2+1x2+1
=11

2進数

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

$2^9 = 512$

$2^{10} = 1{,}024 = 1K$

(参考　1,000=1k)

１６進数、８進数とデジタル

10進	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8進	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24
16進	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14

- 人間はなぜ１０進数を使うか？

  → 手の指が１０本あるから。

- デジタルコンピュータは２進数が基本。

  ではなぜ１６進数、８進数を使うか？

→ ２進数と１６進数、８進数は相性がよいから。

8進数と2進数の変換

8進	2進 (4 2 1)
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

例　8進4桁　3724

●10進に変換

3x8x8x8 + 7x8x8 + 2x8 + 4

計算が必要。

●　2進に変換

011 111 010 100

右表から機械的に得られる。

16進数と2進数の変換

16進	2進	16進	2進
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

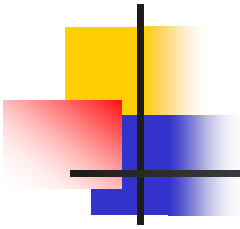
例　16進で3桁

A46

2進数に変換

1010 1000 0110

左表から機械的に得られる。



2進、8進、16進、10進の明確化

例： 1001

2進、8進、16進、10進の区別がつかない

2進	最後に b をつける	1001b	binary
8進	o	1001o	octal
16進	h	1001h	hex
(h の代わりに x を用いることもある)			
10進	d	1001d	decimal

なぜ10月がOctober 12月がDecember？

余談

Oct は8の意味

Dec は10の意味

July (7月）　ローマの英雄　ジュリアス･シーザ

August (8月）　ローマ初代皇帝　アウグスチヌス

が割り込んだため

7月も8月も31日まであるのもこの理由

今から320年前、1692年のパリ

哲学者、数学者、科学者　ライプニッツ

（**Gottfried Wilhelm Leibniz**）

「全ての数を1と0によって表す驚くべき表記法」

を提案。

王立科学アカデミーに理解されず

学会誌にも掲載されなかった。

「誰も予想しなかった卓越した用途がありはずだ」

と語る。（慶応義塾大学　青山先生資料）

ゴットフリート・ヴィルヘルム・ライプニッツ（Gottfried Wilhelm Leibniz, 1646年 – 1716年）

ライプニッツは哲学者、数学者、科学者など幅広い分野で活躍した学者・思想家として知られているが、また政治家であり、外交官でもあった。17世紀の様々な学問（法学、政治学、歴史学、神学、哲学、数学、経済学、自然哲学（物理学）、論理学等）を統一し、体系化しようとした。その業績は法典改革、モナド論、微積分法、微積分記号の考案、論理計算の創始、ベルリン科学アカデミーの創設等、多岐にわたる。ライプニッツは稀代の知的巨人といえる。

デジタル・コンピュータとプログラミング

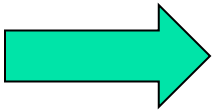
デジタル・コンピュータで仕事をさせうるには
全てを指示してやらなければならない（プログラミング）

- 理工系大学院生の問題を解くのは得意

例： 連立3次元偏微分方程式を
境界条件のもとに数値計算で解く

- 人間の赤ちゃんの問題を解くのは苦手

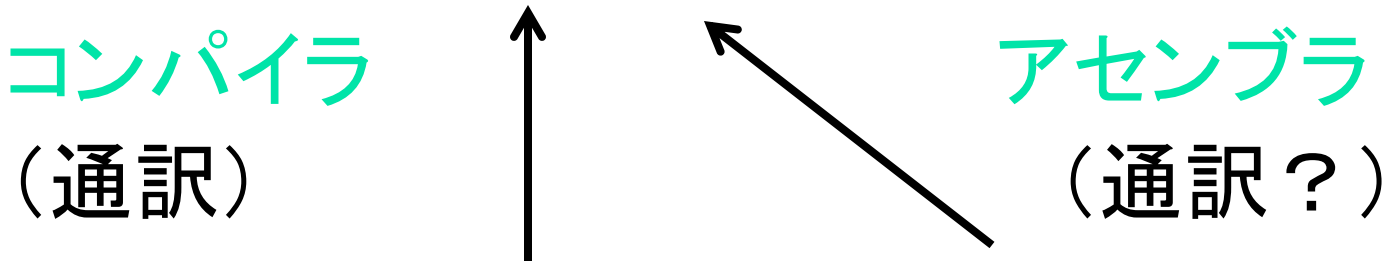
例： お母さんの顔を認識する



プログラミングが大変

高級言語、アセンブラ言語、機械語

DSPチップ　　機械語(0,1)　東京標準語



プログラマ　　高級言語　　アセンブラ言語
（人間）　　（C言語等）　　大阪弁
英語

高級言語、アセンブラ言語、機械語 （2）

- アセンブラ言語のほうが高級言語より
  よいプログラム（高速、小容量）がかける。
- 大阪弁を東京標準語に通訳（？）する方が
  英語を　　”　　より容易。
- 現実のプログラム開発
  大部分は高級言語で記述。
  どうしても高速化・小容量化したい部分は
  アセンブラ言語で記述。

C言語とアセンブラ言語

- C言語は一種類(“方言”少ない)。

  どのコンピュータでも動作する。

  コンピュータ内部の構成と動作を知らなくてもプログラミングできる。

- アセンブラ言語はプロセッサ毎に異なる。

  コンピュータ内部の構成と動作を知らないと

  プログラミングできない。

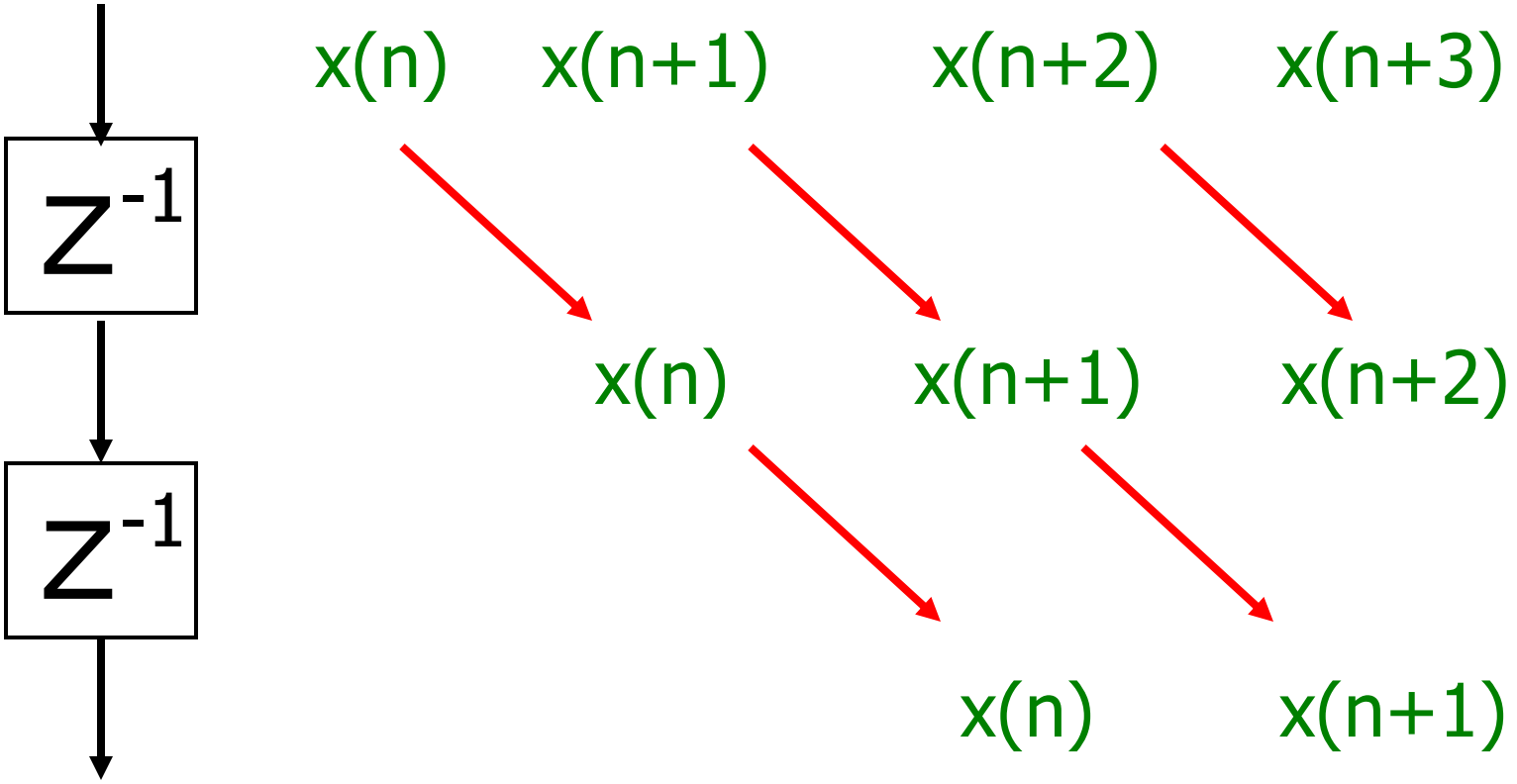
  アセンブラ言語によるプログラミングは

  「組み込みソフトウェア」に関連しても重要な技術

アセンブラ命令

LT　Y0　　：Load T-register
MPY H0　　：Multiply
PAC　　：P-register ⇒ ACC
APAC　　：ACC + P-register ⇒ ACC
SACH　*+,1　：Store ACC High（間接アドレッシング）
SACH　Y0,1　：Store ACC High (直接アドレッシング）
DMOV　　：Data Move
B　Loop　　：Branch (分岐）
LDP　#Y0　　：Load DP-Register

シフト演算子 z^{-1}



.ds 0F00h

データメモリ

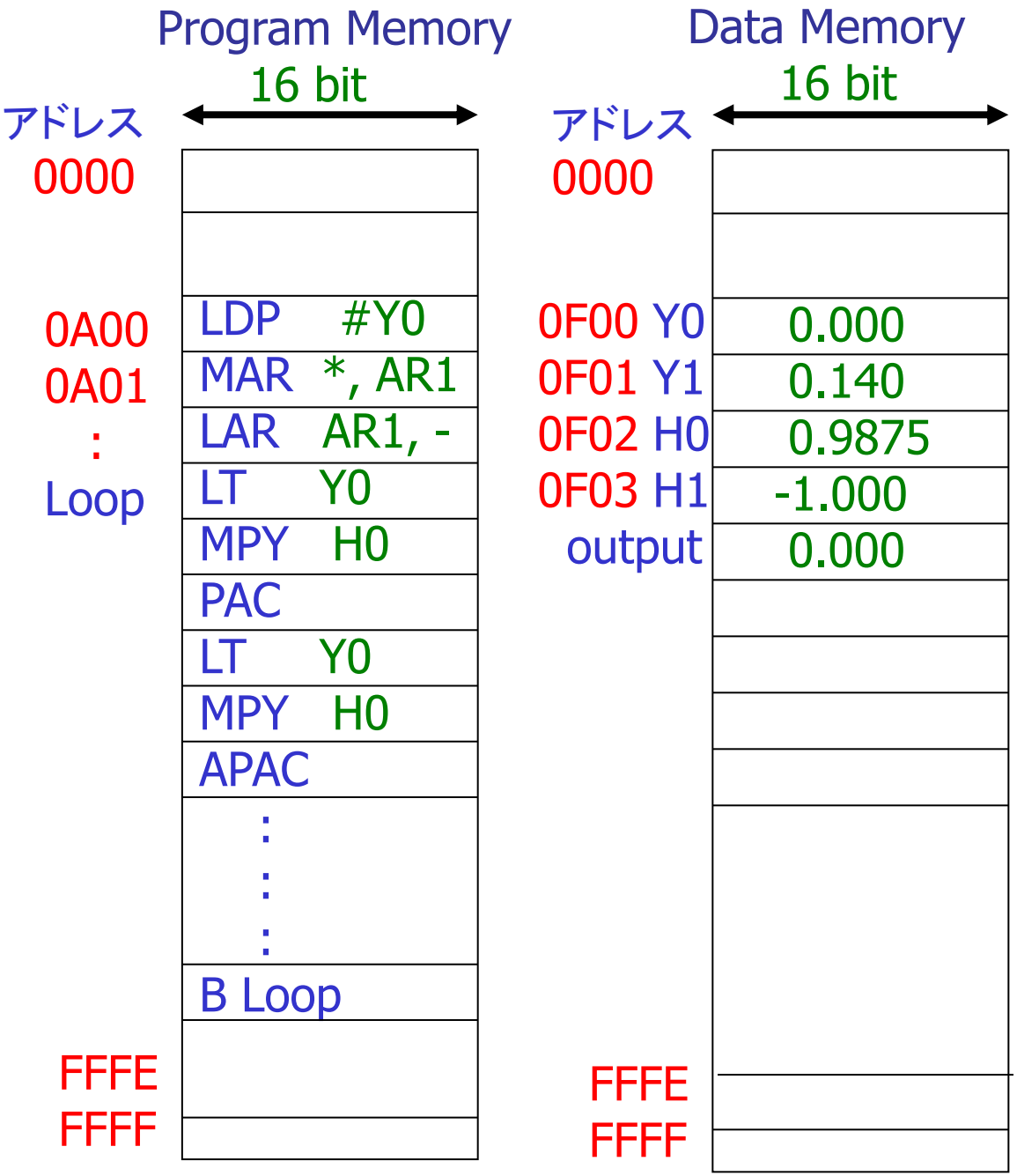
0F00番地から
以下のデータを
おく、の意味。

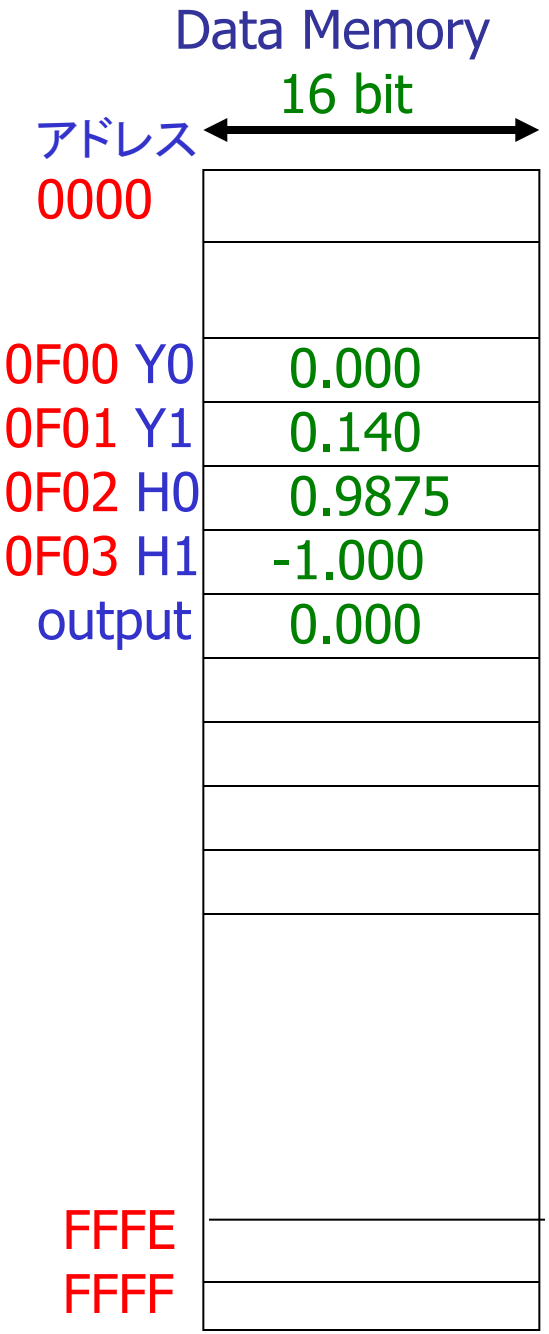
.ps 0A00h

プログラムメモリ

0A00番地から
以下のプログラム
をおく、の意味。

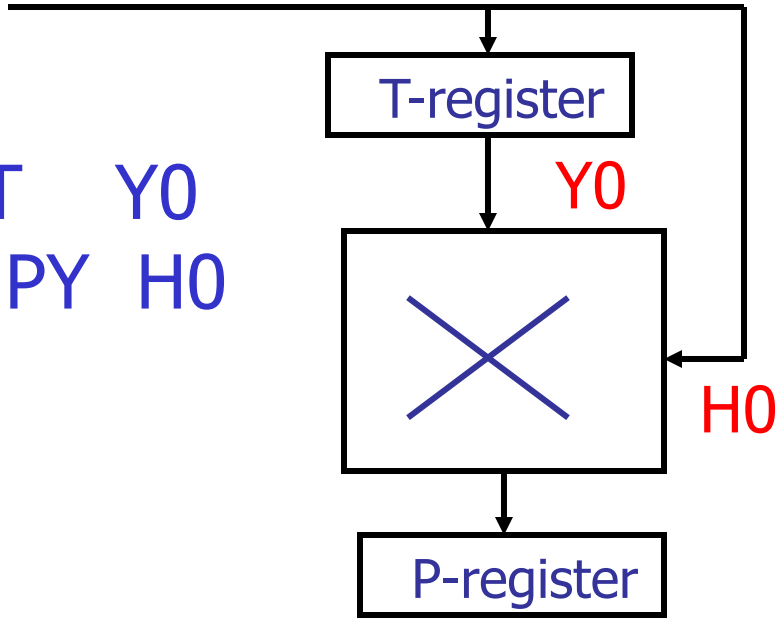
h は16進数
(hexagonal)
の意味。



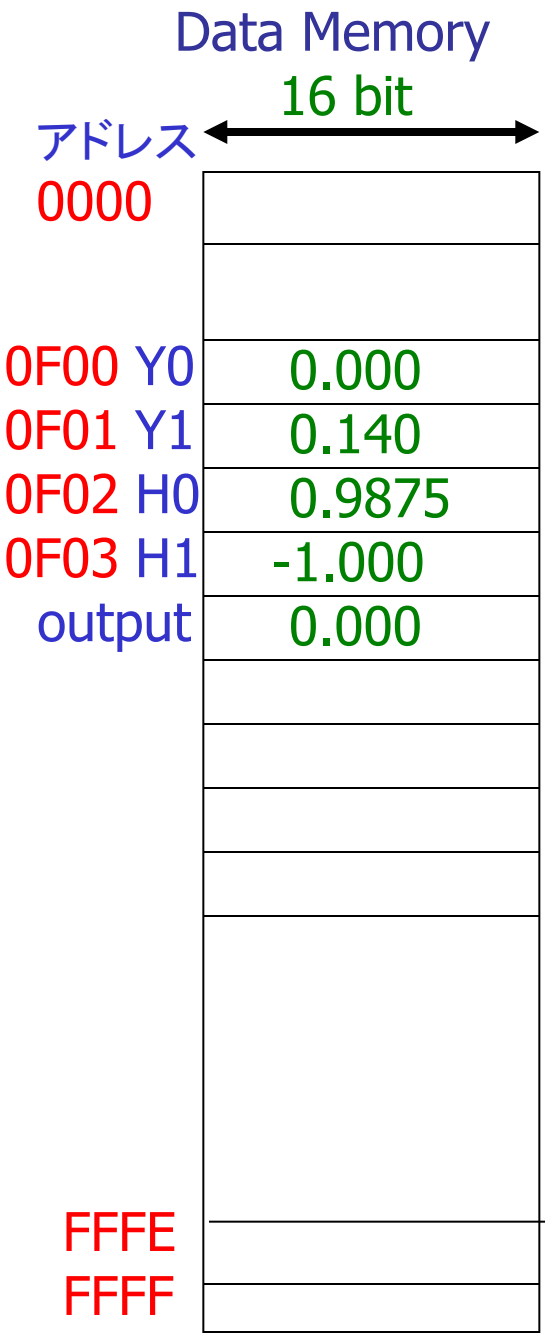


$$Y = \boxed{H0 \cdot Y0} + H0 \cdot Y0 + H1 \cdot Y1$$

LT Y0
MPY H0

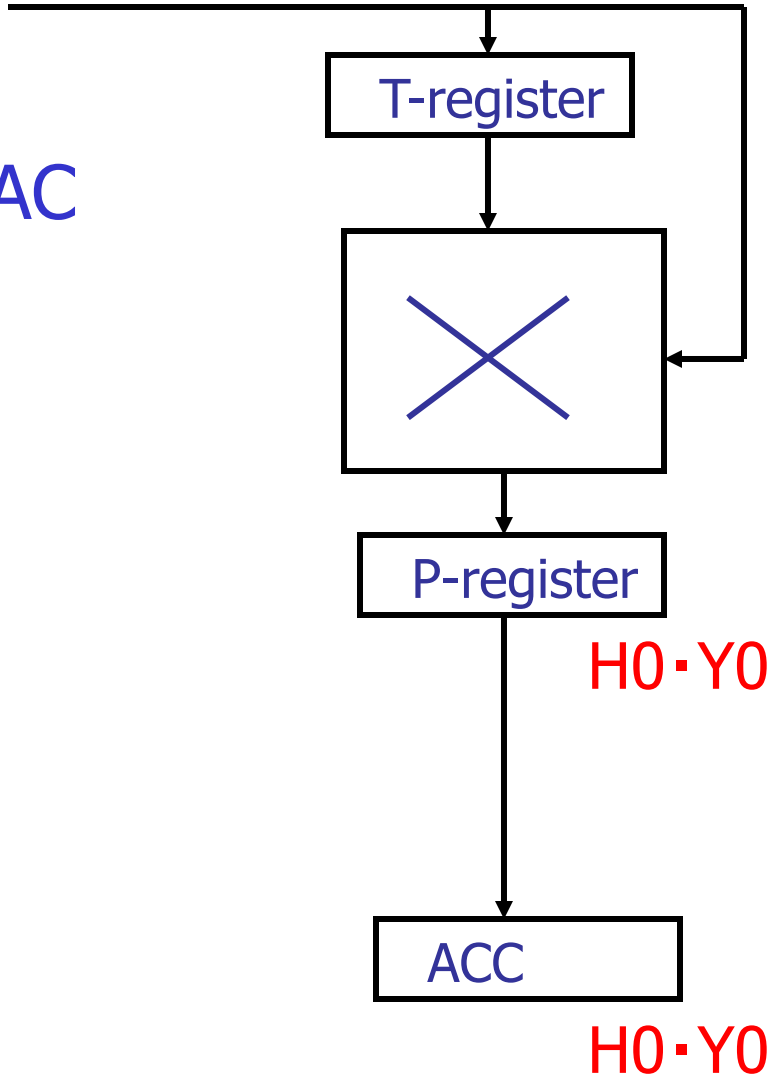


H0·Y0



$$Y = H0 \cdot Y0 + H0 \cdot Y0 + H1 \cdot Y1$$

PAC



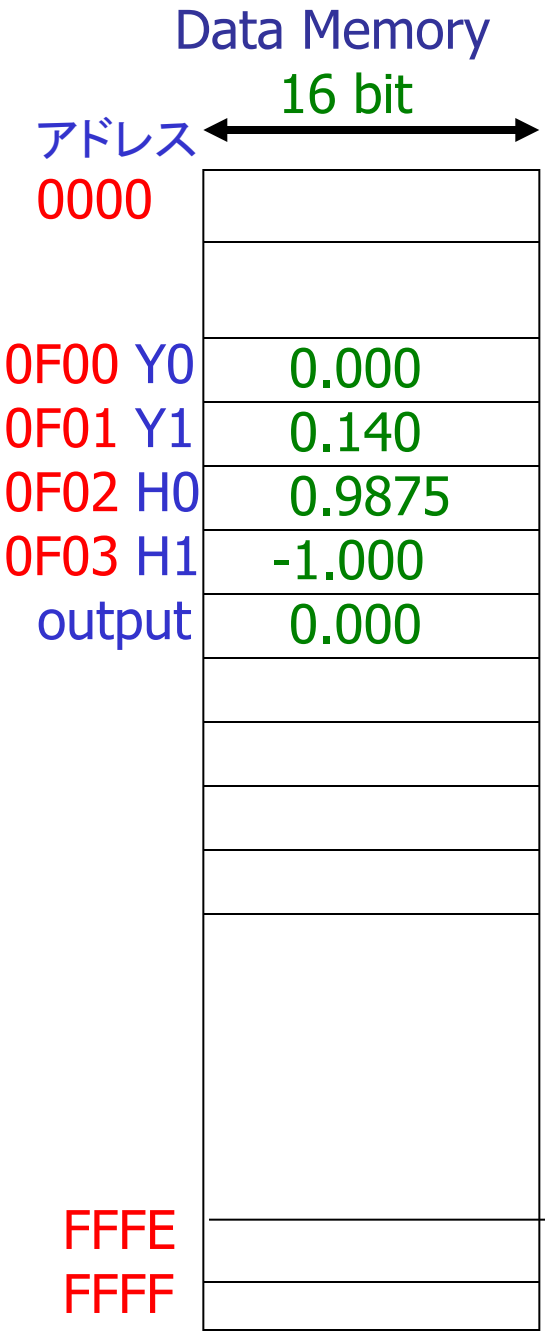
四則演算の英語での表現

＋　　add

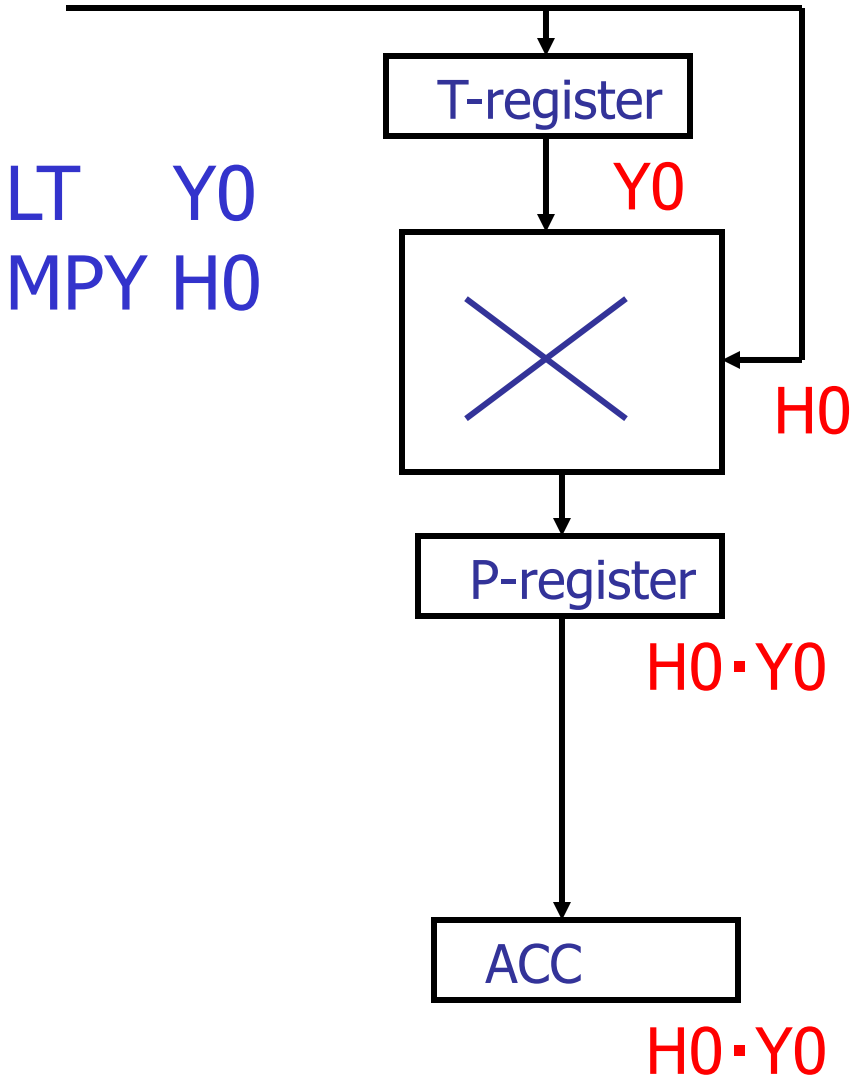
－　　subtract

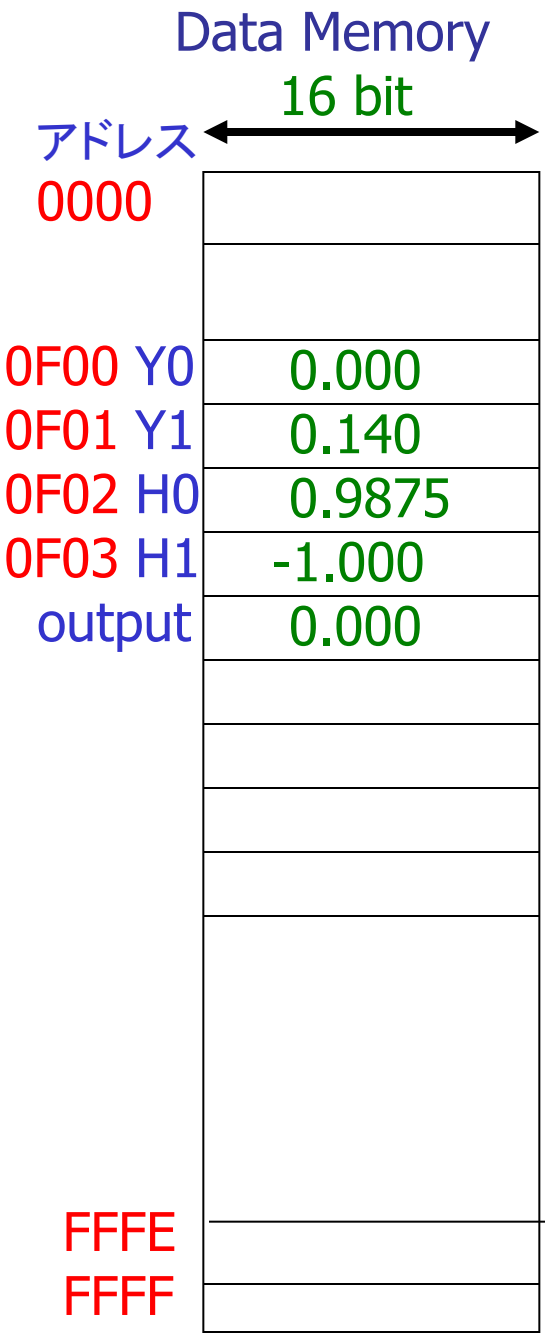
X　　multiply

÷　　divide

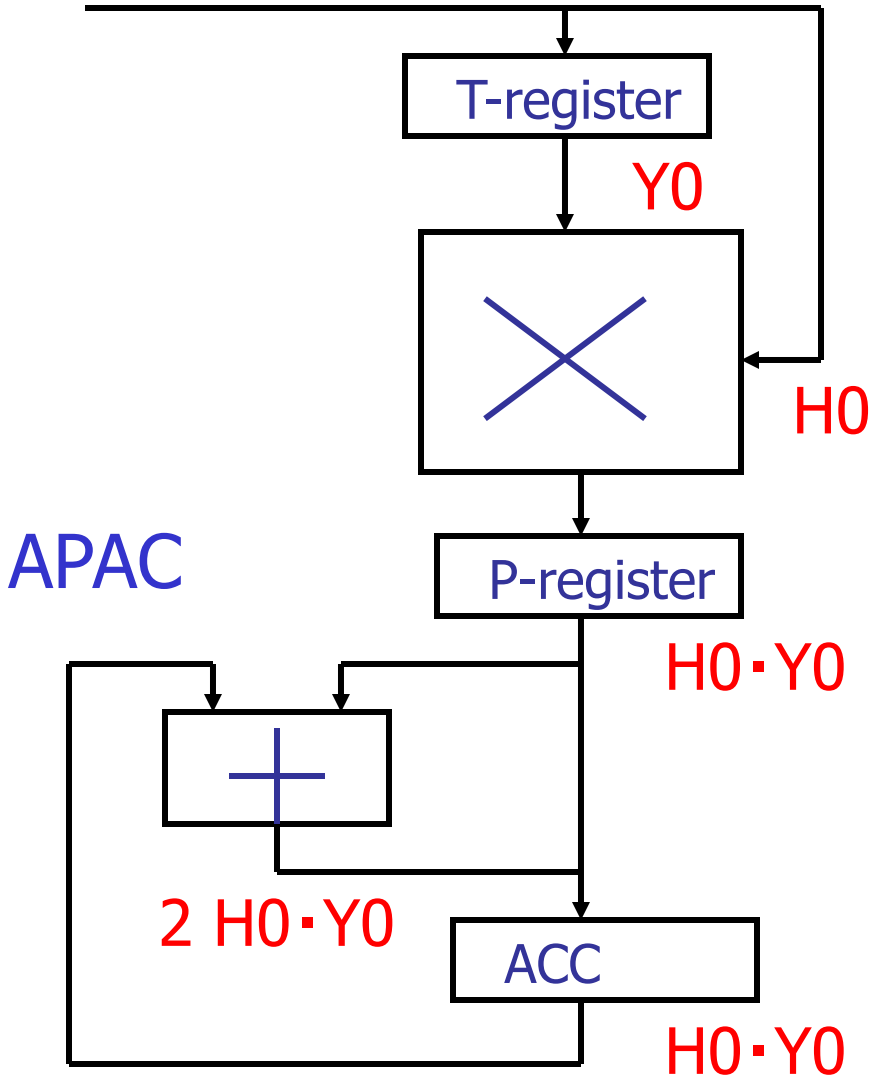


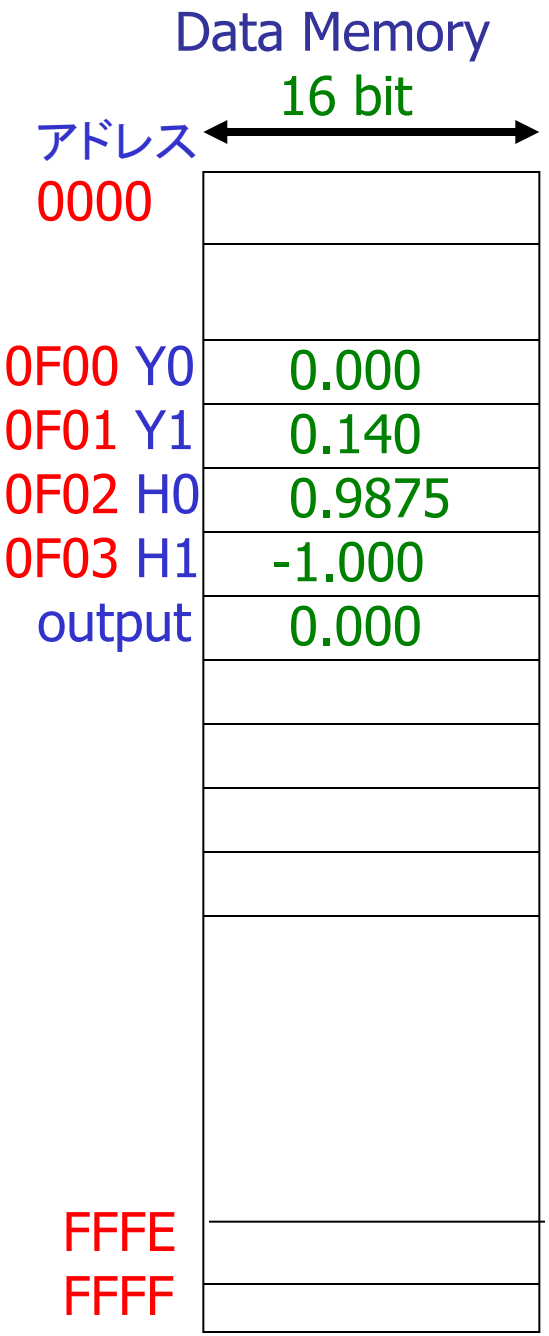
$$Y = H0 \cdot Y0 + \boxed{H0 \cdot Y0} + H1 \cdot Y1$$





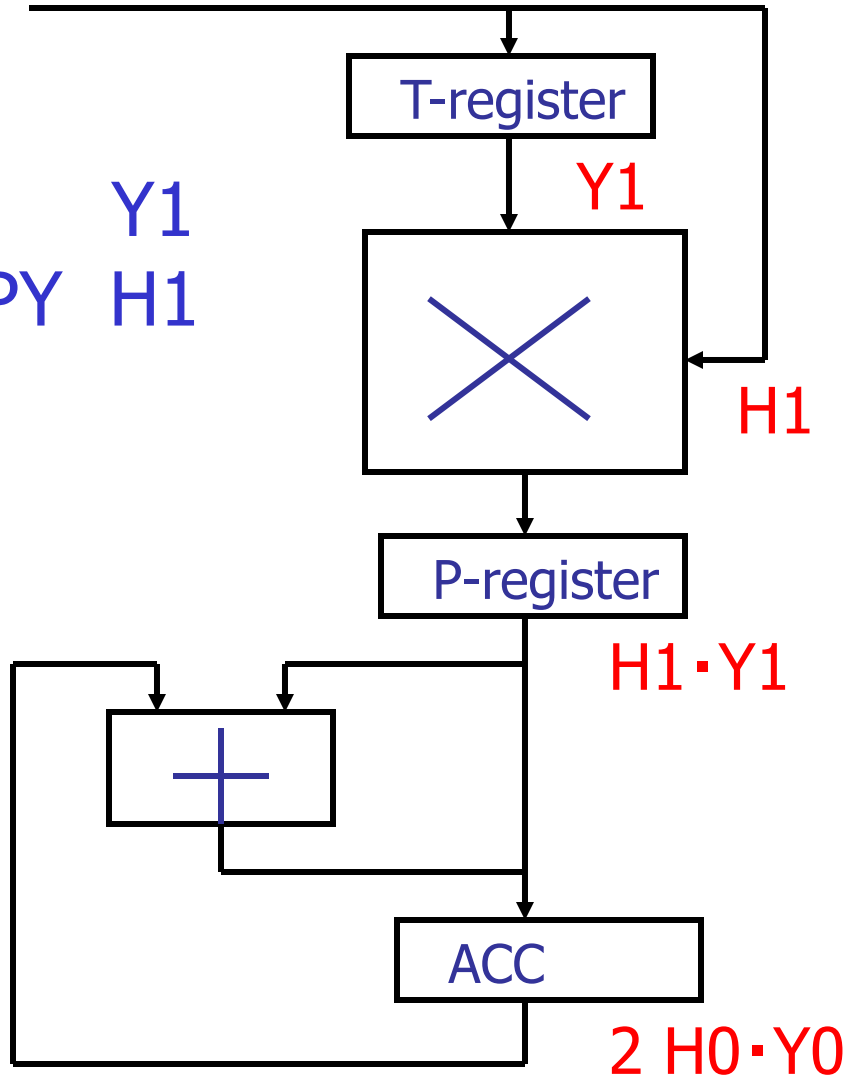
$$Y = H0 \cdot Y0 + H0 \cdot Y0 + H1 \cdot Y1$$

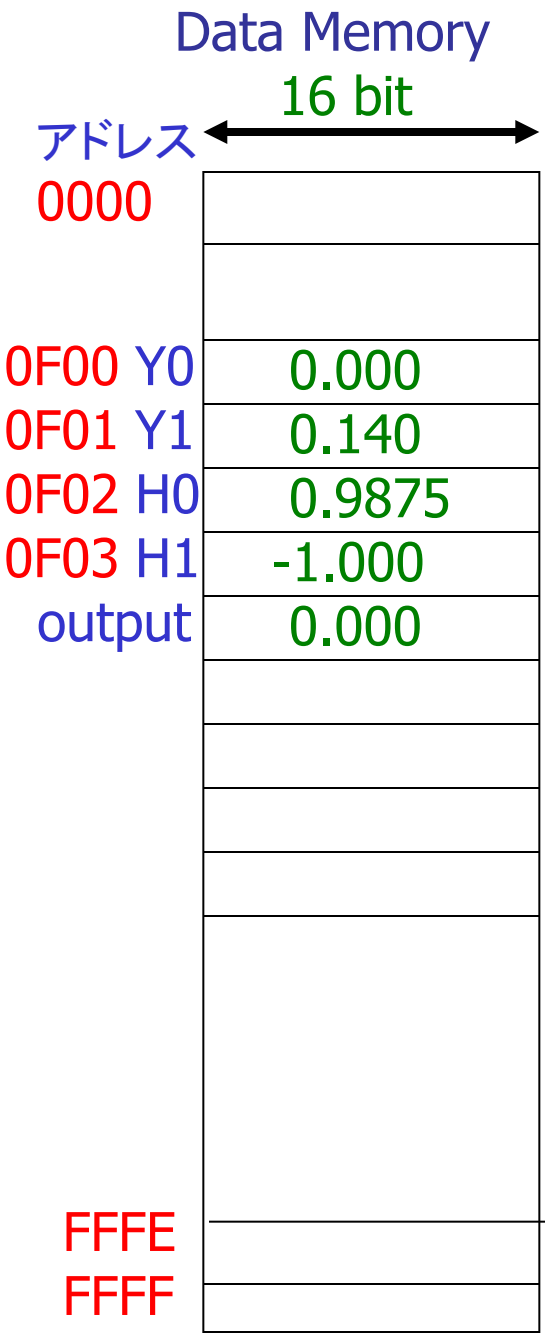




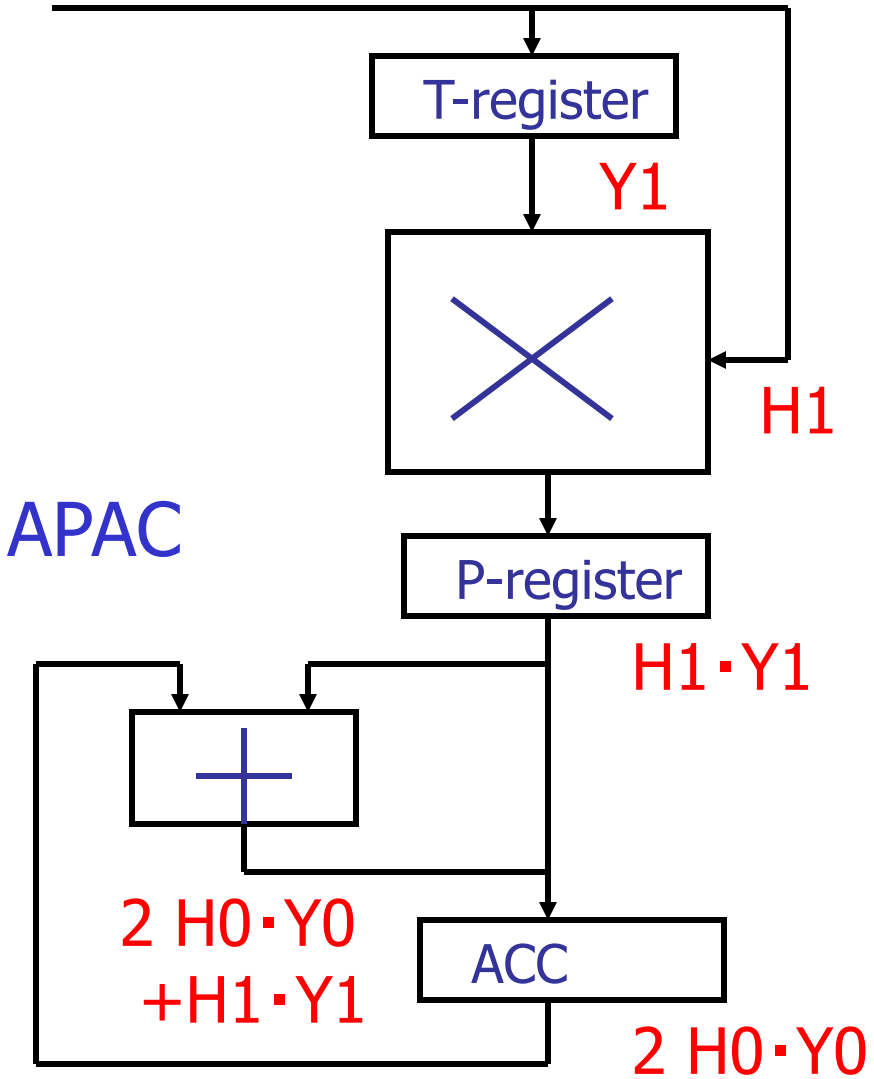
$$Y = H0 \cdot Y0 + H0 \cdot Y0 + H1 \cdot Y1$$

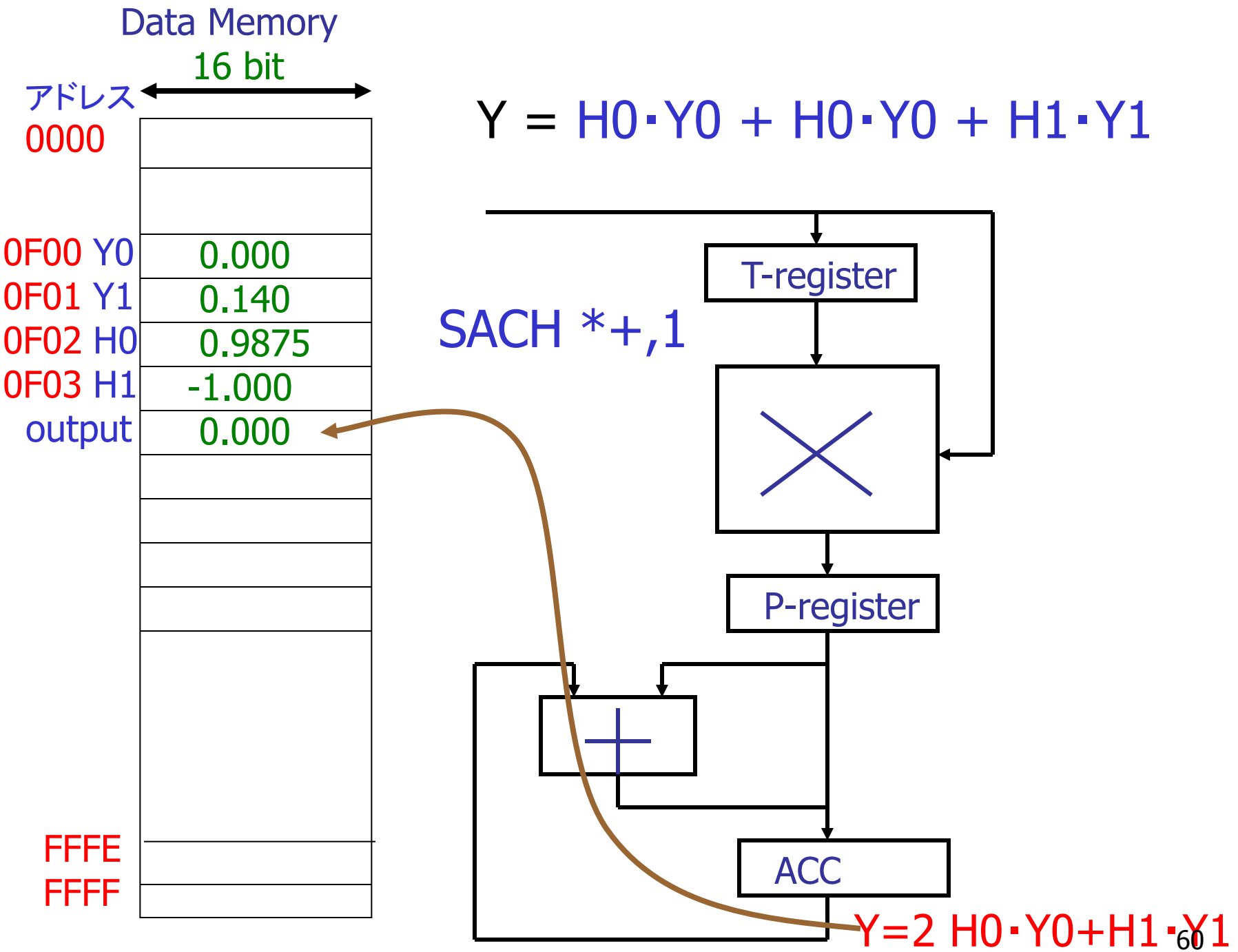
LT Y1
MPY H1

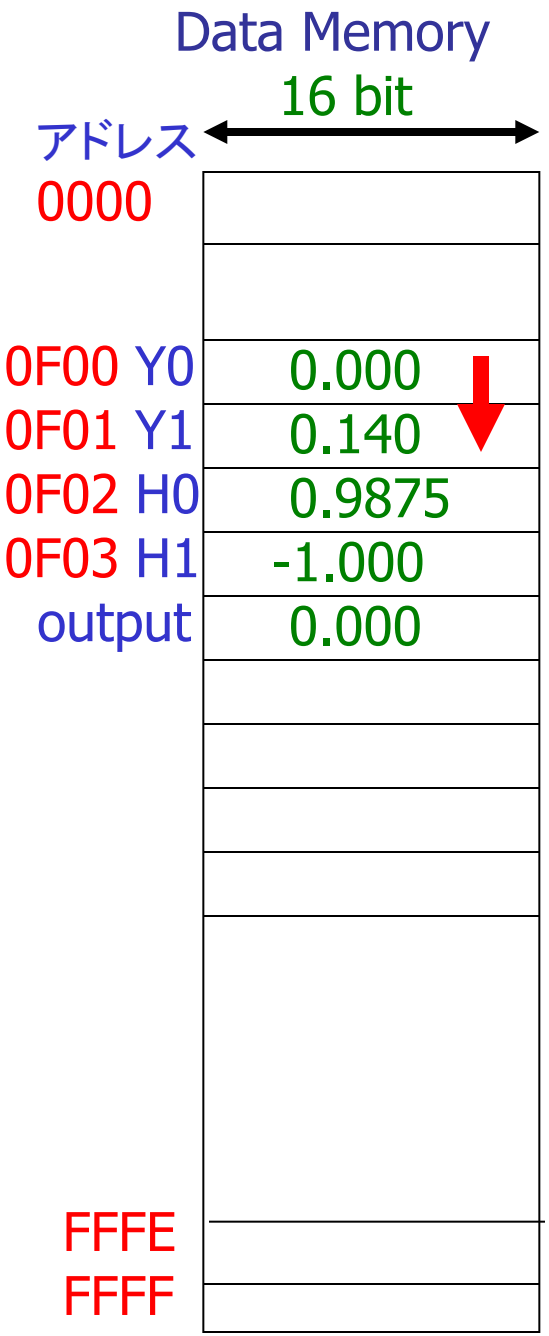




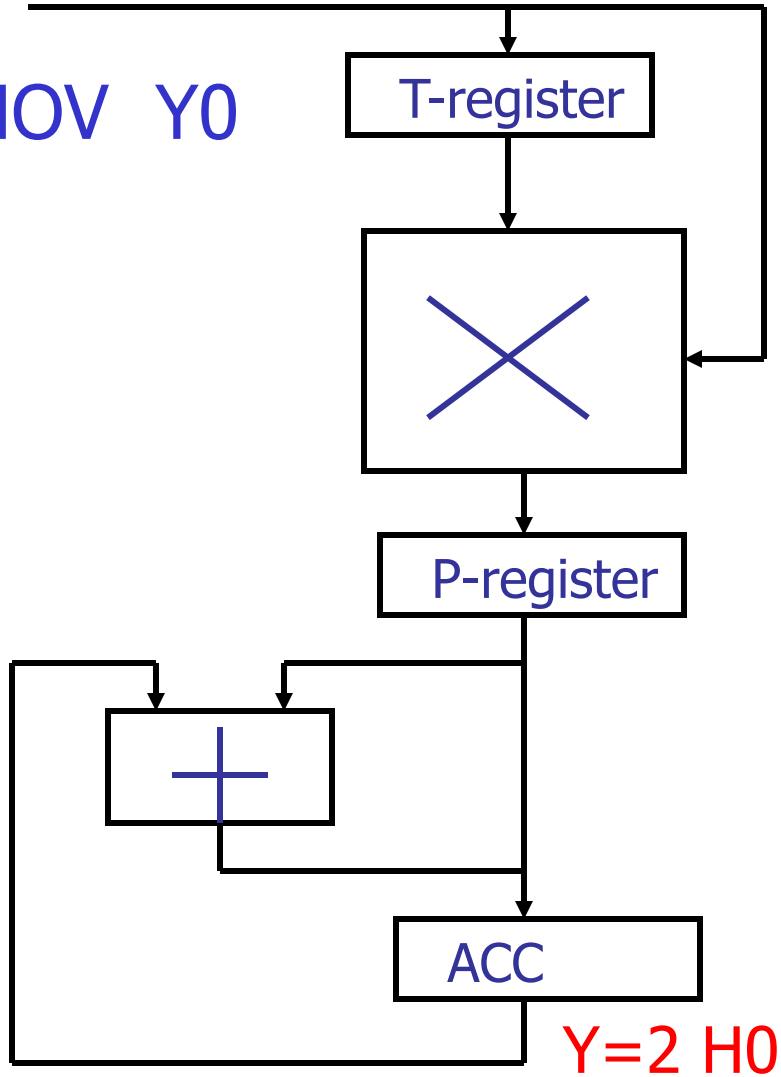
$$Y = H0 \cdot Y0 + H0 \cdot Y0 + H1 \cdot Y1$$



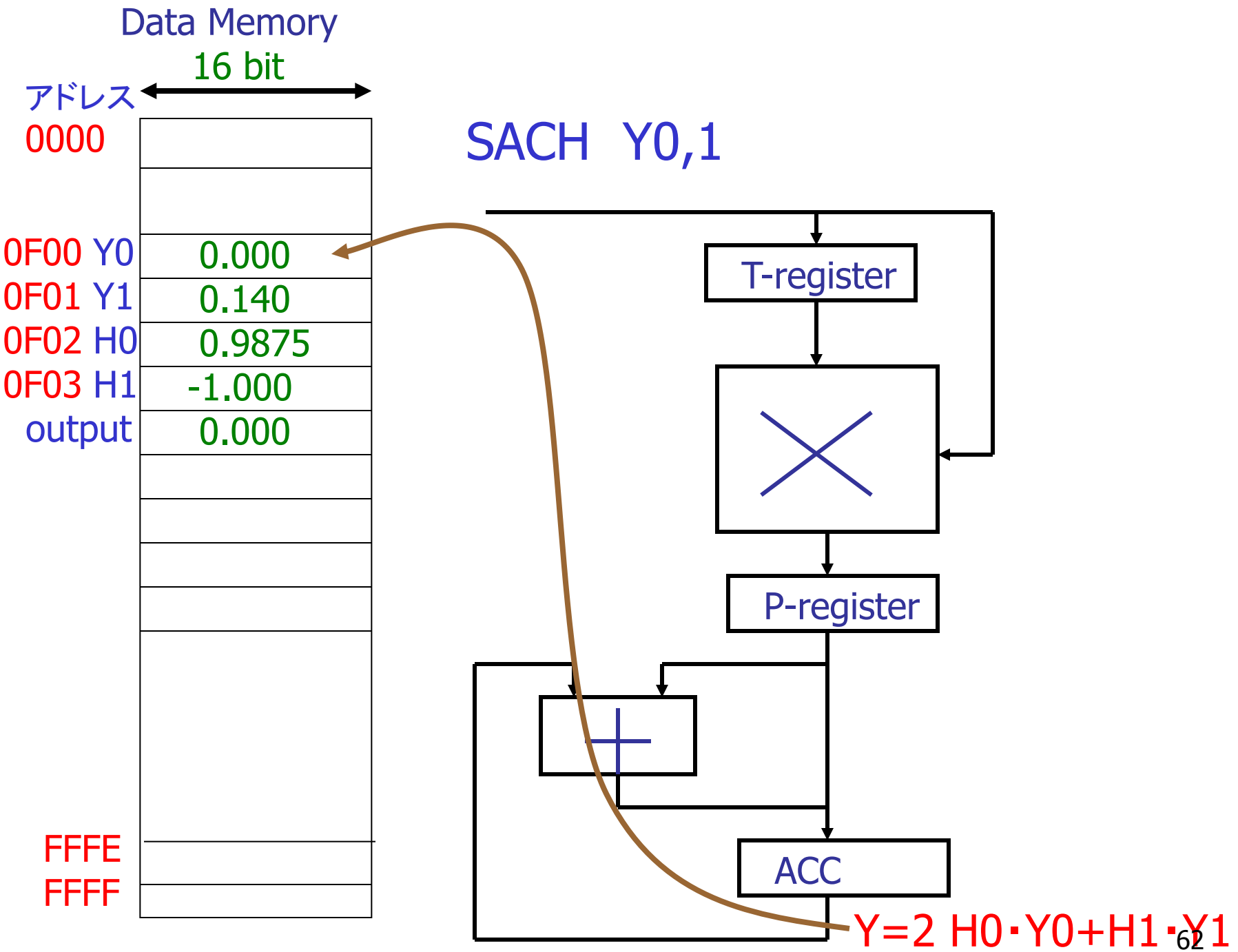

Data Memory
16 bit
アドレス
0000
0F00 Y0
0F01 Y1
0F02 H0
0F03 H1
output
FFFE
FFFF
0.000
0.140
0.9875
-1.000
0.000
Y = H0·Y0 + H0·Y0 + H1·Y1
SACH *+,1
T-register
P-register
ACC
Y=2 H0·Y0+H1·Y1



DMOV Y0



$Y=2\ H0 \cdot Y0+H1 \cdot Y1$


Data Memory
16 bit
アドレス
0000
0F00 Y0
0.000
0F01 Y1
0.140
0F02 H0
0.9875
0F03 H1
-1.000
output
0.000
FFFE
FFFF
SACH Y0,1
T-register
P-register
ACC
Y=2 H0・Y0+H1・Y1

LDP 命令

Y0（アドレス）

0F00 （16進数）

0000 1111 0000 0000 （2進数）

0000 LT

0001 MPY

0010 PAC

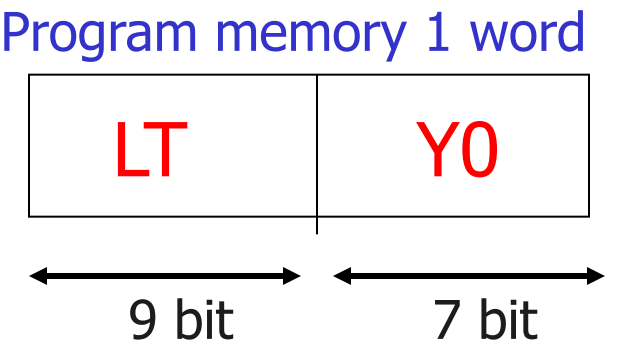
0011 APAC

0100 SACH

0101 DMOV

0110 B

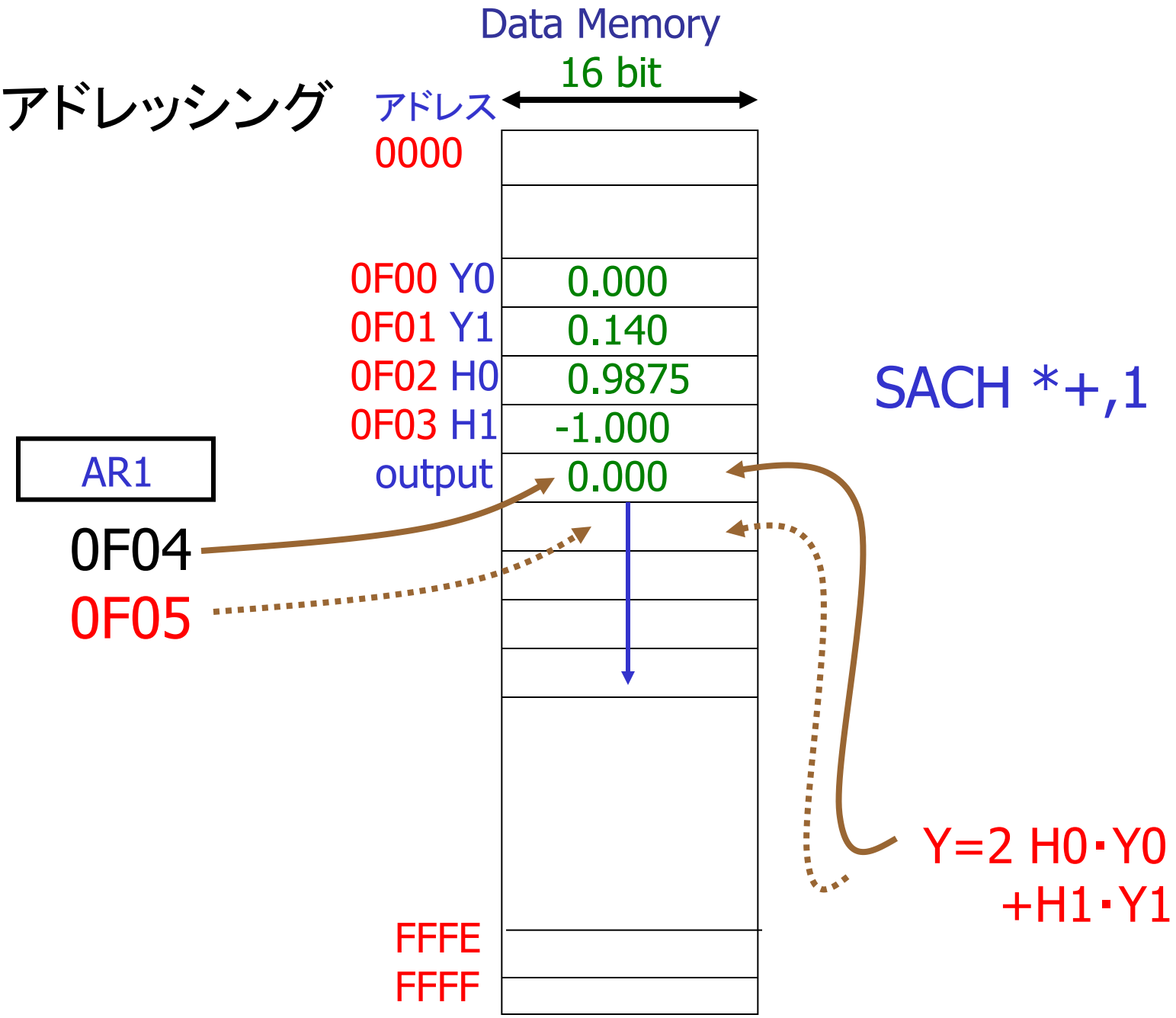
0111 LDP

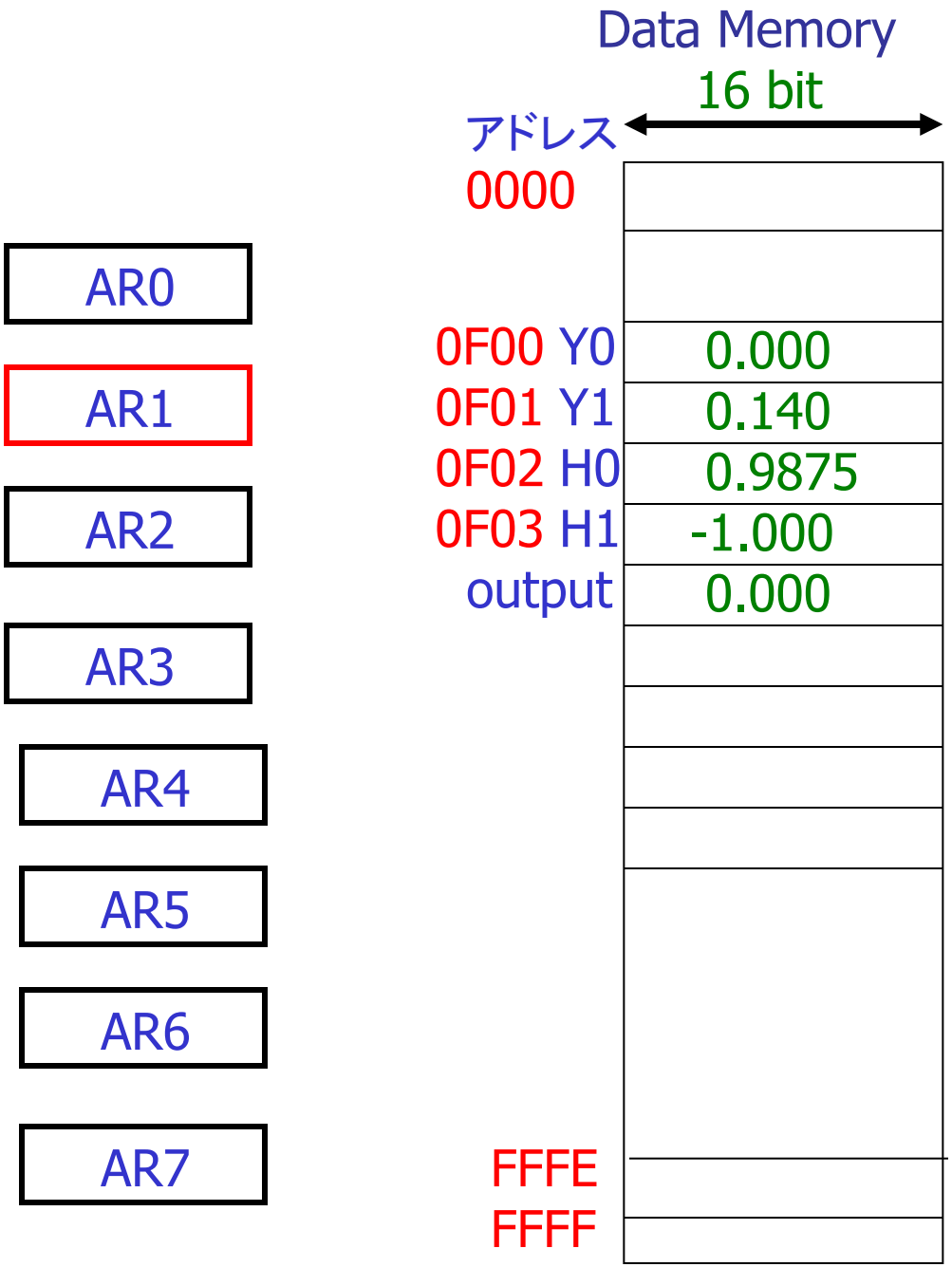


000 0000

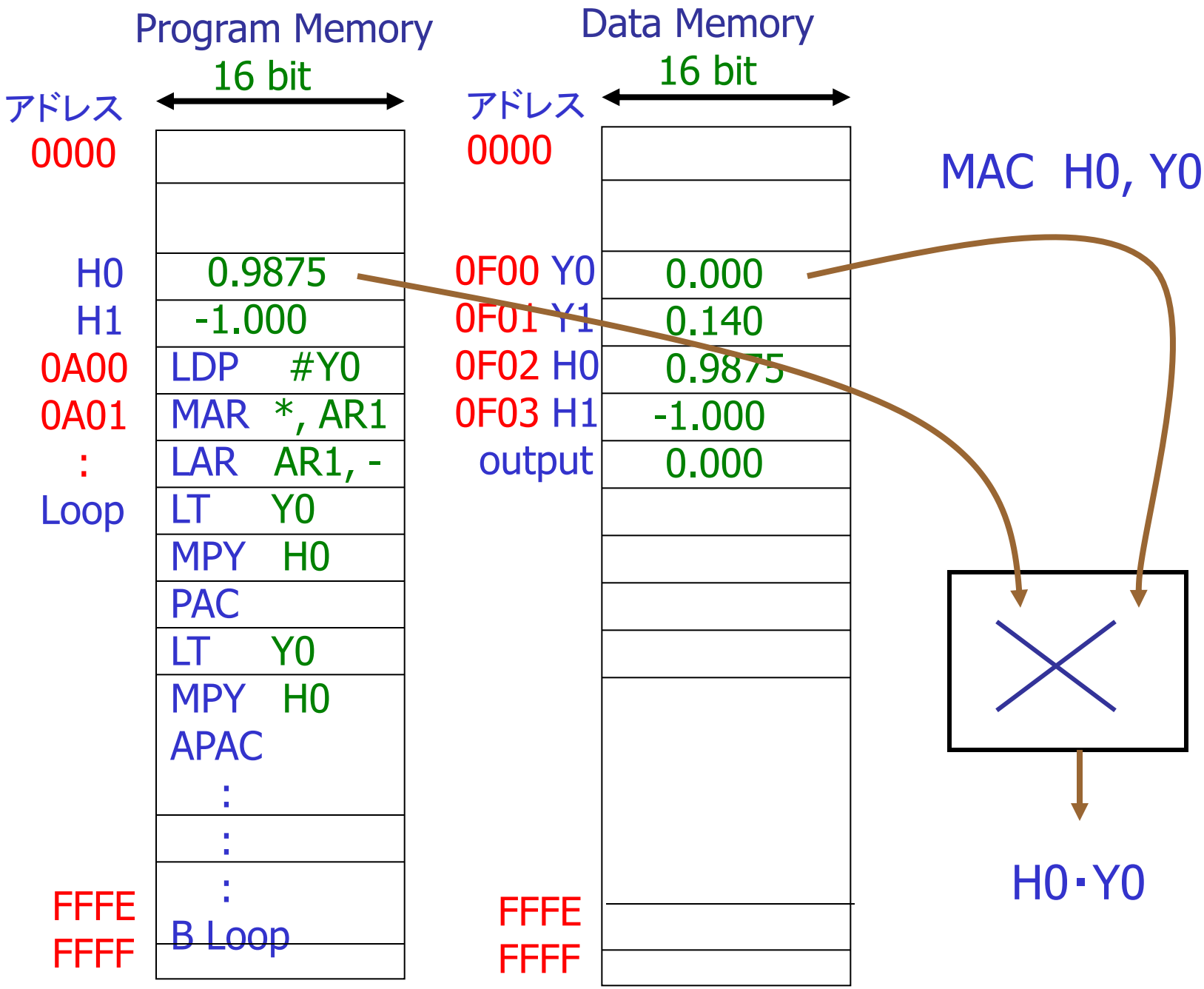
0000 1111 0

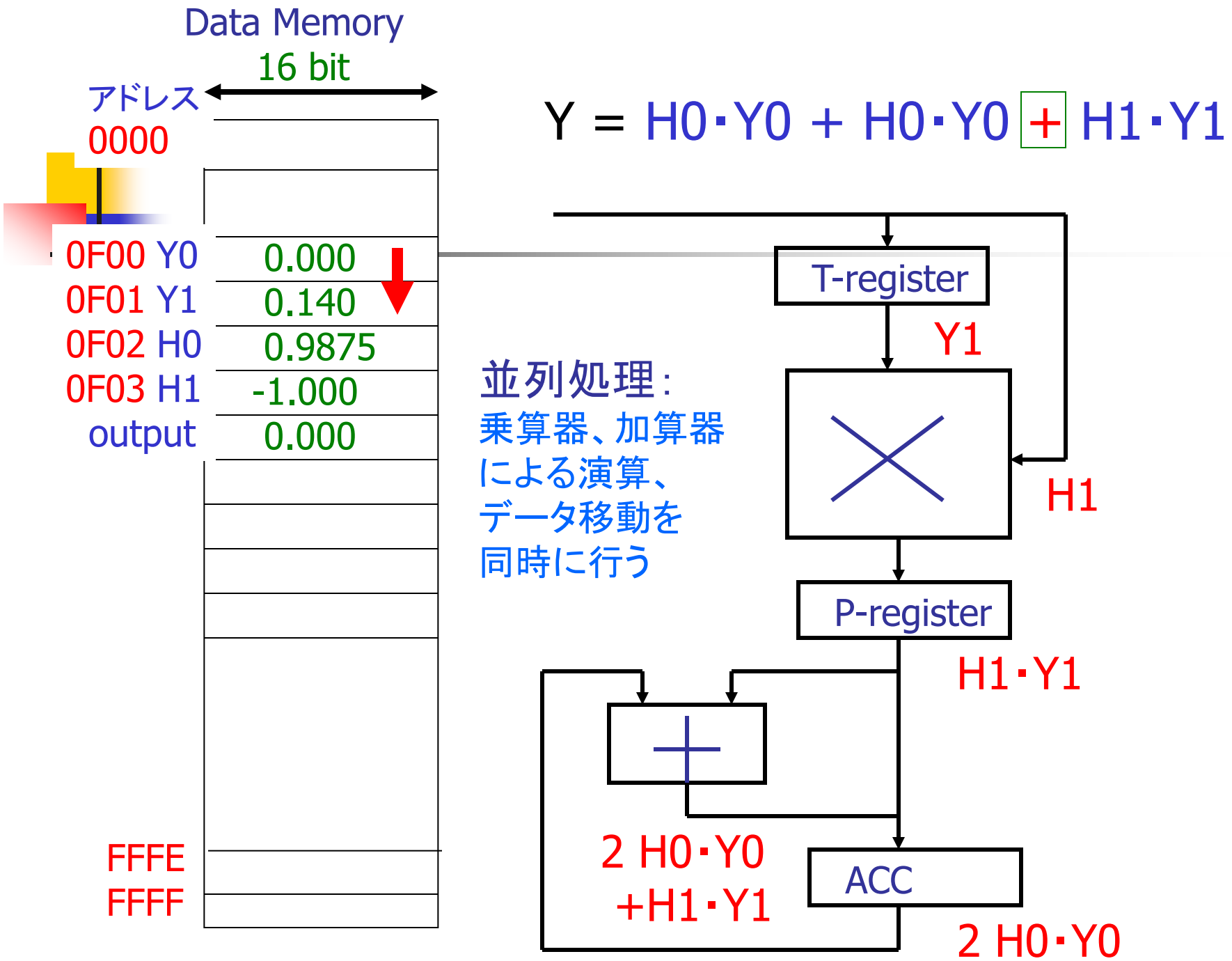
間接アドレッシング

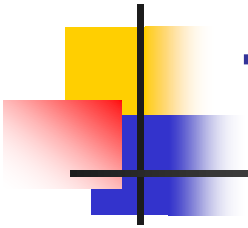

Data Memory
16 bit
アドレス
0000
0F00 Y0
0.000
0F01 Y1
0.140
0F02 H0
0.9875
0F03 H1
-1.000
output
0.000
AR1
0F04
0F05
SACH *+,1
Y=2 H0･Y0
+H1･Y1
FFFE
FFFF


Data Memory
16 bit
アドレス
0000
AR0
AR1
AR2
AR3
AR4
AR5
AR6
AR7
0F00 Y0
0F01 Y1
0F02 H0
0F03 H1
output
0.000
0.140
0.9875
-1.000
0.000
FFFE
FFFF

MAR *, AR1


Program Memory
16 bit
アドレス
0000
H0
H1
0A00
0A01
:
Loop
FFFE
FFFF
0.9875
-1.000
LDP #Y0
MAR *, AR1
LAR AR1, -
LT Y0
MPY H0
PAC
LT Y0
MPY H0
APAC
:
:
:
B Loop
Data Memory
16 bit
アドレス
0000
0F00 Y0
0F01 Y1
0F02 H0
0F03 H1
output
FFFE
FFFF
0.000
0.140
0.9875
-1.000
0.000
MAC H0, Y0
H0・Y0


Data Memory
16 bit
アドレス
0000
0F00 Y0
0F01 Y1
0F02 H0
0F03 H1
output
0.000
0.140
0.9875
-1.000
0.000
FFFE
FFFF
Y = H0･Y0 + H0･Y0 + H1･Y1
並列処理:
乗算器、加算器
による演算、
データ移動を
同時に行う
T-register
Y1
H1
P-register
H1･Y1
2 H0･Y0
+H1･Y1
ACC
2 H0･Y0



プログラミングと水泳

「プログラミング」はやってみないとわからない。
本を読み講義を聴いただけではわからない。

本を読み　話しをきいただけでは
泳げるようにならないとと同じ。

プログラミングは特にその色彩がつよい。

DSPの研究者、研究開発拠点

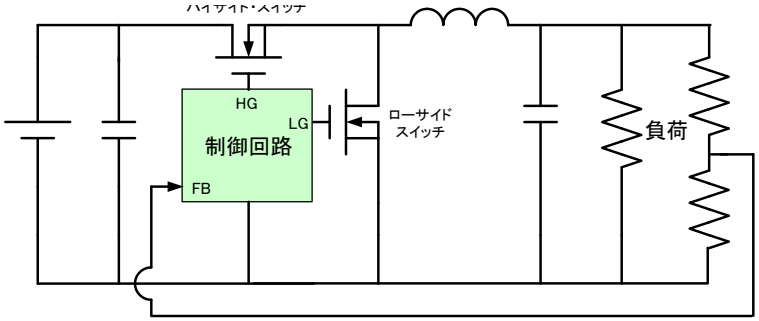
- MIT Prof. A. Oppenheim
  DSP の神様、テキストはベストセラー
- UCLA Prof. H. Samueli (Broadcom 創業者）
  アルゴリズムに加えて“IC化”の技術
- Georgia Institute of Technology（米　アトランタ）
  多くのDSP研究者
- ベルギー　ルーベン市
  DSP Valley,　IMEC, KUL, Target Compiler Technologies
- テキサスインスツルメンツ社、アナログデバイセズ社
  DSPとアナログ

最近の話題：　電源もDSPで制御

デジタル制御電源

コスト･電力の課題はあるがデジタル化の流れ

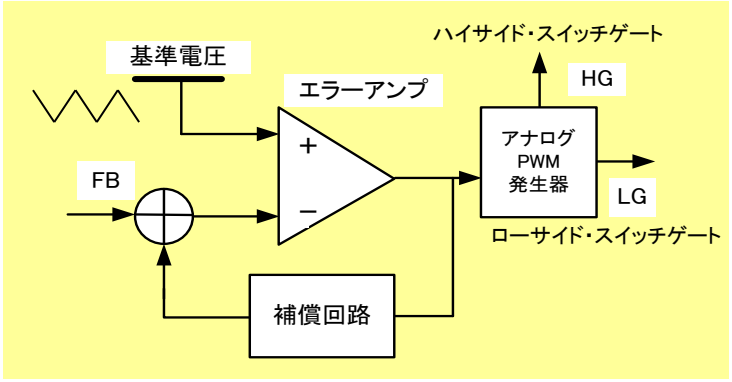
- スイッチング電源回路



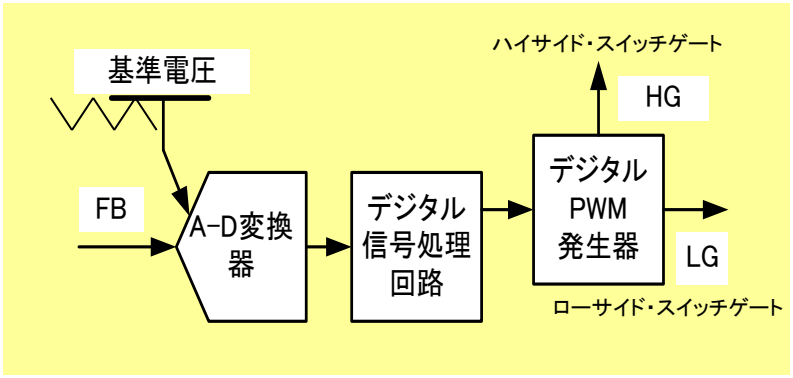
- 外資系半導体メーカー
  パワーマネージメント製品に注力
- 微細CMOSでデジタル制御
- デジタルの新アイデアで高性能化
- 通信機能の取り込み

- 制御回路部

■アナログ方式

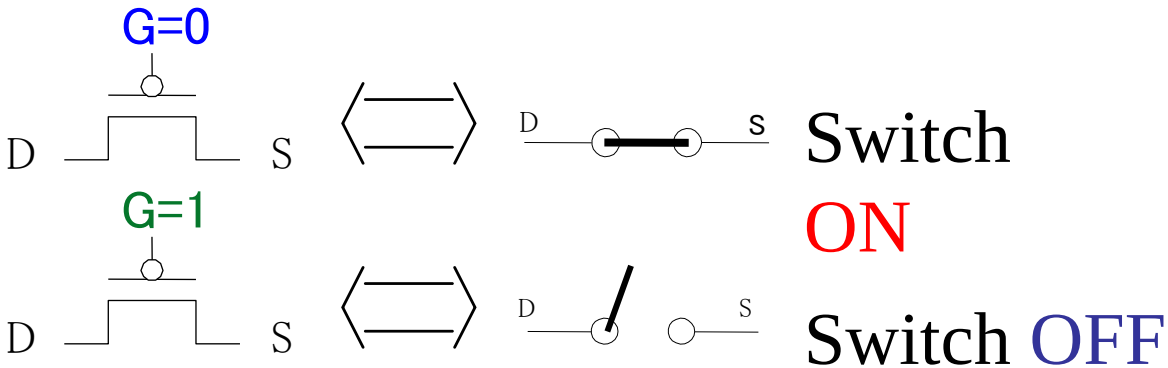


■デジタル方式

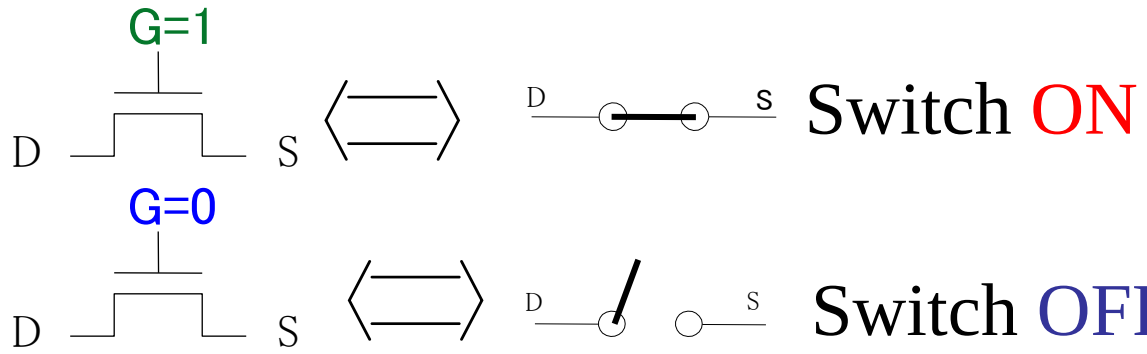


PMOS，NMOS スイッチ

（1）PMOS

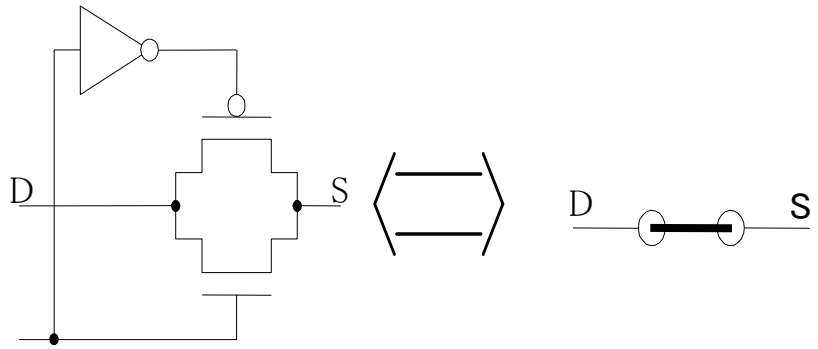


（2）NMOS



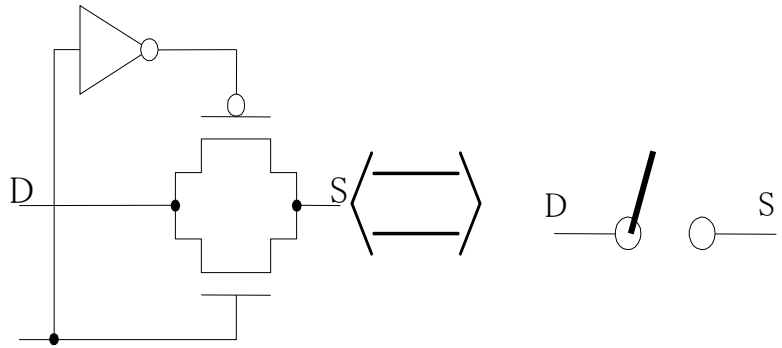
CMOSスイッチ

(3) CMOS



G=1

Switch ON

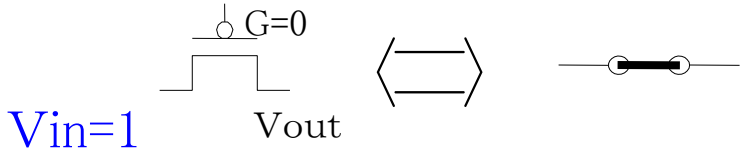


G=0

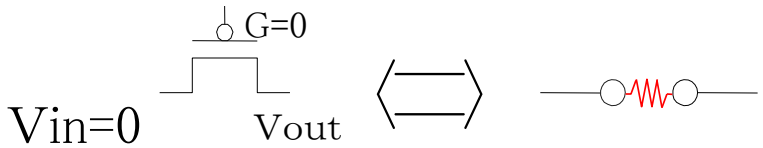
Switch OFF

PMOS，NMOSスイッチのオン抵抗

(1) PMOS



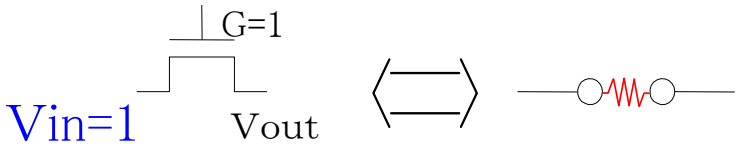
Small
ON-Resistance



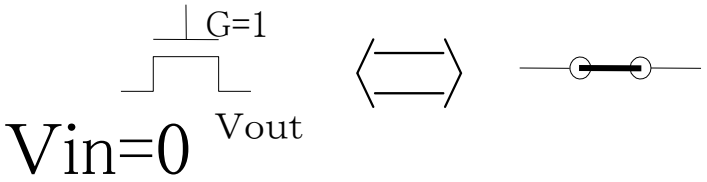
Large
ON-Resistance

PMOSは
正電源側で
用いる

(2) NMOS



Large
ON-Resistance

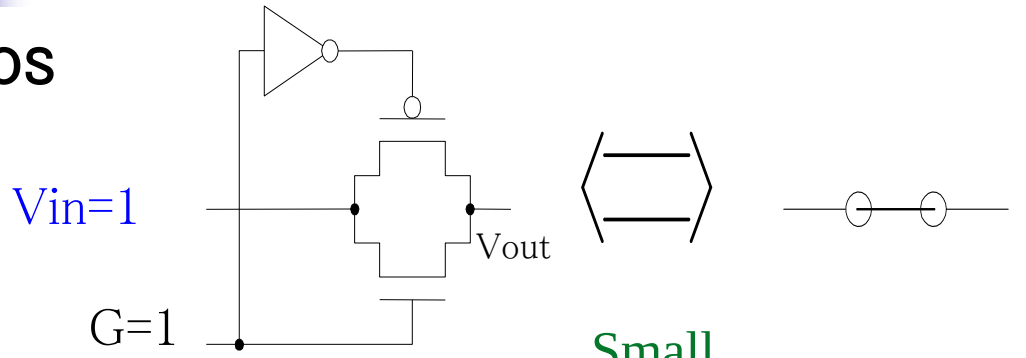


Small
ON-Resistance

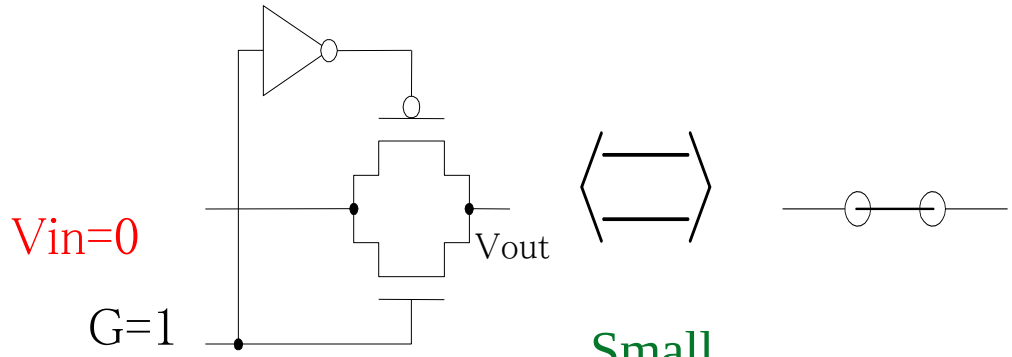
NMOSは
GND側で
用いる

CMOSスイッチのオン抵抗

(3) CMOS



Small
ON-Resistance



Small
ON-Resistance

CMOSは
GND側でも
正電源側でも
オン抵抗が
小さいが、
トランジスタ数
が増える。

論理否定(NOT)

論理変数　A, Z

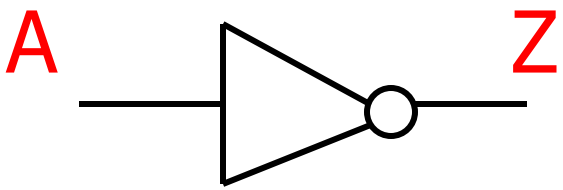
A:入力,　Z:出力

$Z = \overline{A}$

真理値表

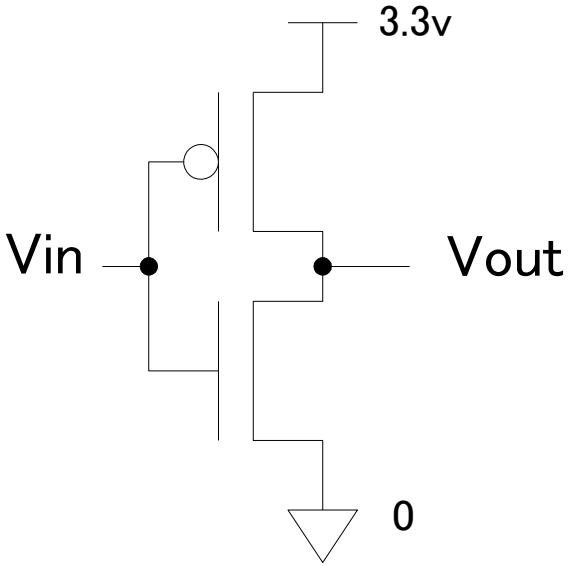
A	Z
0	1
1	0

NOT を実現する回路 ⇒ インバータ回路

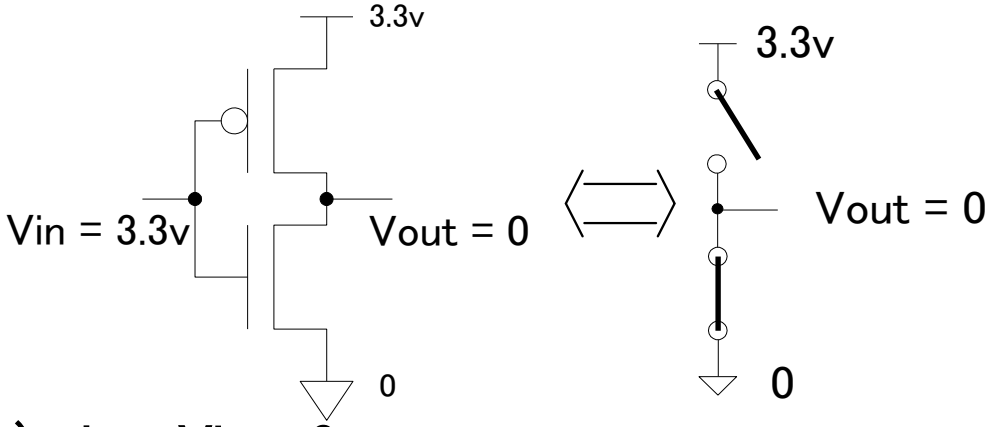


CMOSインバータ回路

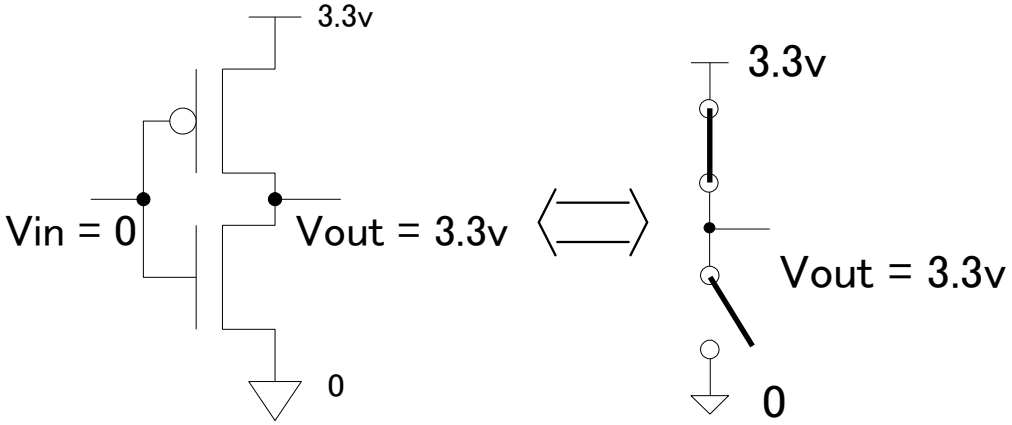
Inverter



a) when Vin = 1 (3.3v)



b) when Vin = 0



NAND　(NAND = AND + NOT)

論理変数　A,B, Z

A,B：入力，　Z：出力

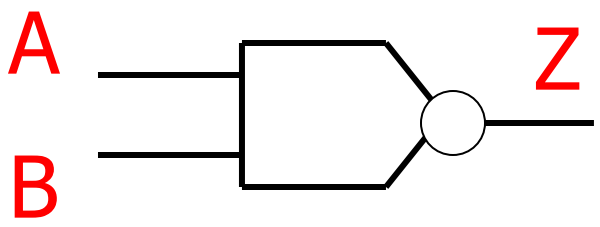
$$Z = \overline{A \cdot B}$$

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

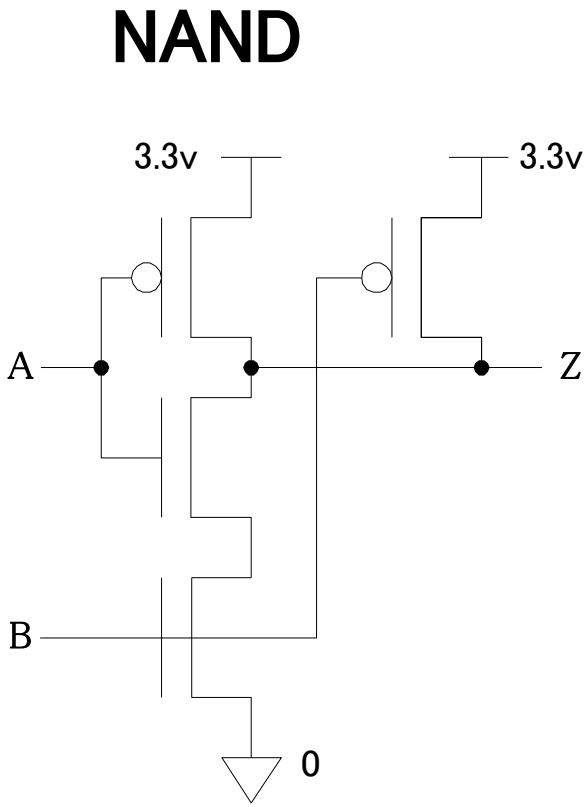
真理値表

NANDを実現する回路

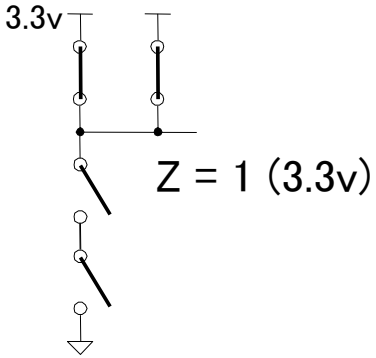
NAND回路



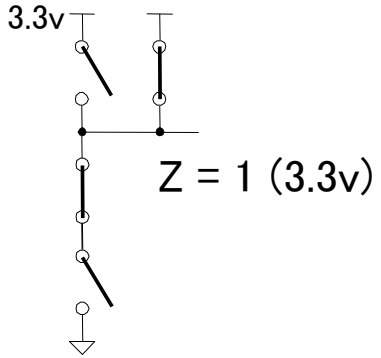
CMOS NAND回路



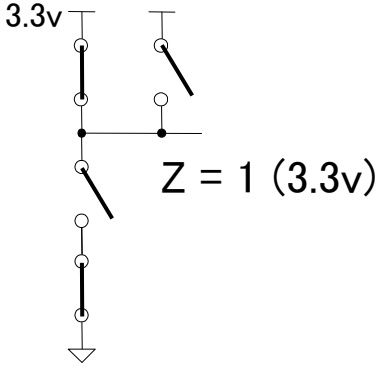
a) when A=0, B=0



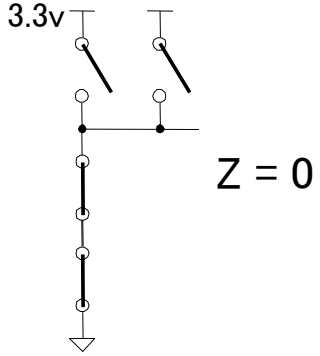
b) when A=1, B=0



c) when A=0, B=1



d) when A=1, B=1



NOR　（NOR = OR + NOT）

論理変数　A,B, Z

A,B：入力，　Z：出力

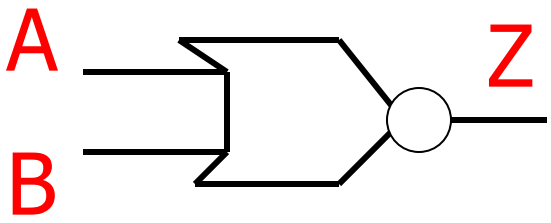
$$Z = \overline{A+B}$$

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

真理値表

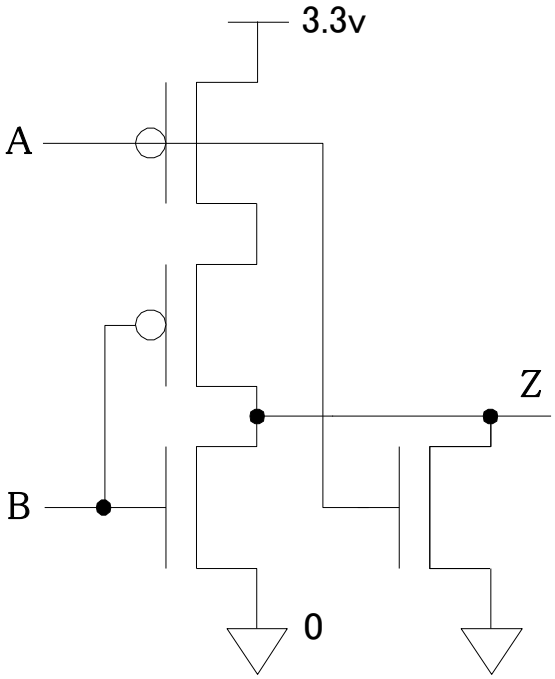
NORを実現する回路

⇒ NOR回路

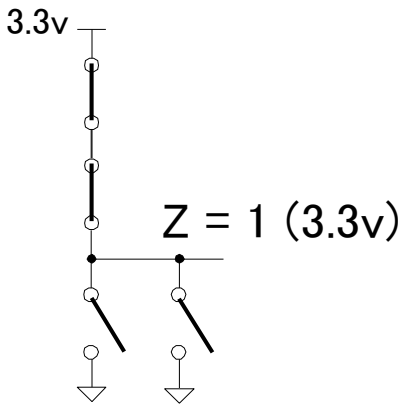


CMOS NOR回路

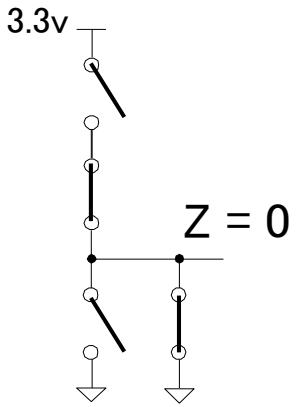
NOR 回路



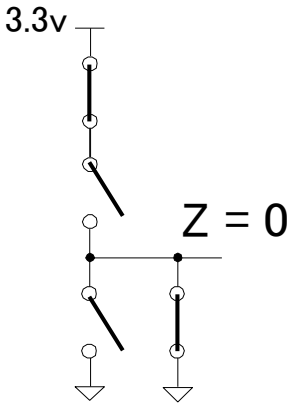
a) when A=0, B=0



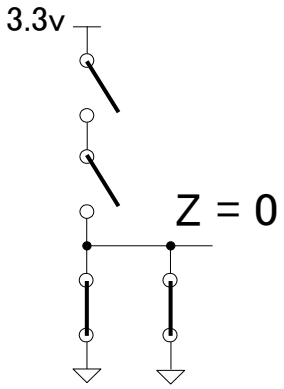
b) when A=1, B=0



c) when A=0, B=1



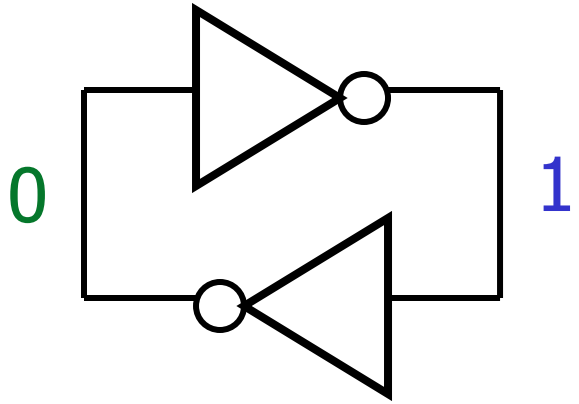
d) when A=1, B=1



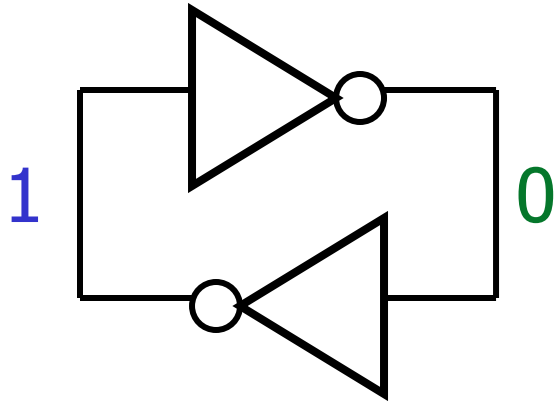
2つのインバータのリング接続
メモリ回路

2つの安定状態

データ“1”を記憶



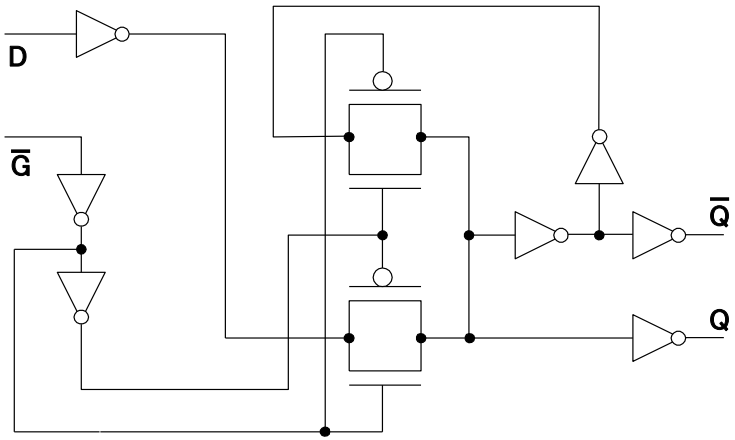
データ“0”を記憶



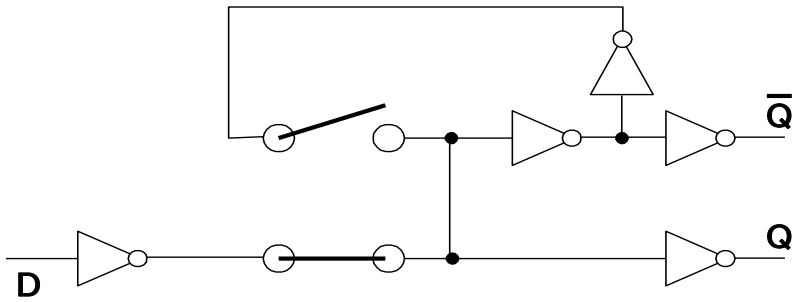
- SRAM (Static ランダム・アクセス・メモリ)
  Latch, Flip-Flop 等のメモリ素子は
  これを利用している。

CMOS ラッチ回路

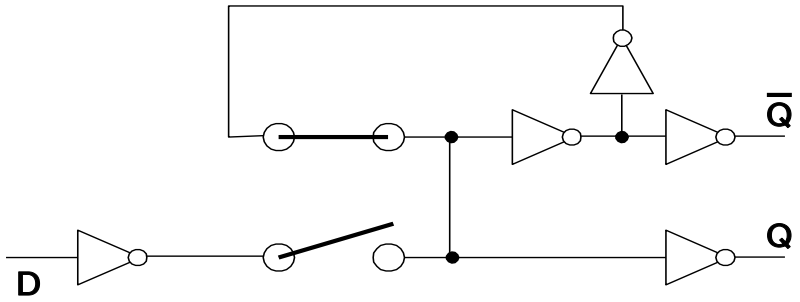
Latch回路
（メモリ素子）



a) when $\overline{G} = 0$



b) when $\overline{G} = 1$



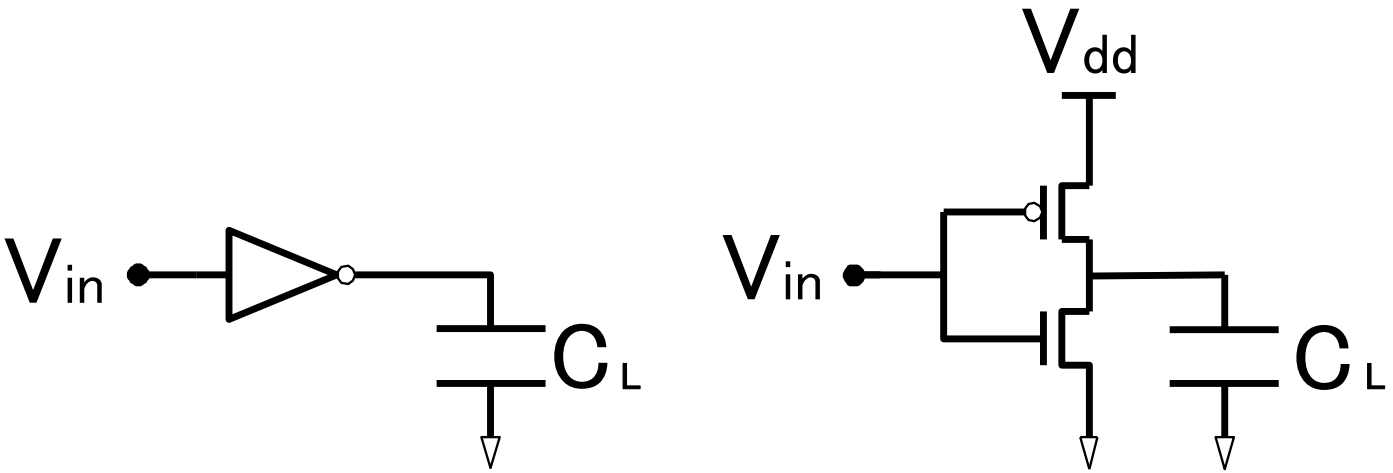
デジタルCMOS回路の電力消費

デジタルCMOS回路（インバータ）

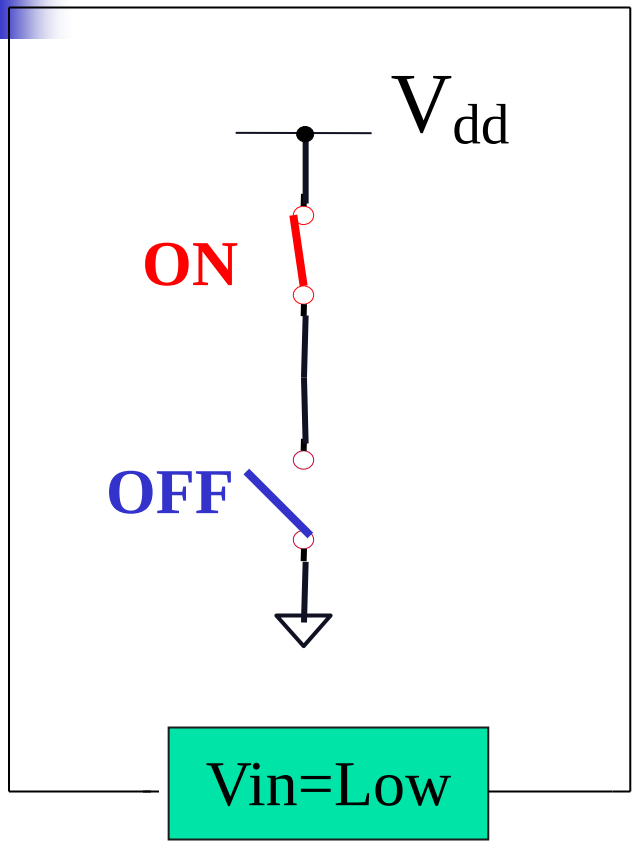
Vdd: 電源電圧

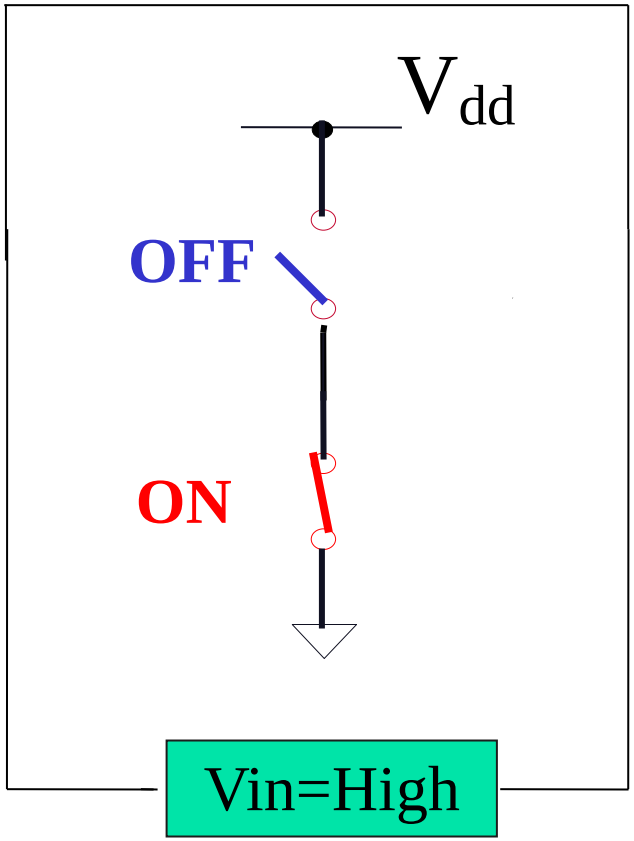
Vin: 入力、　Vout: 出力

C_L ： 負荷容量



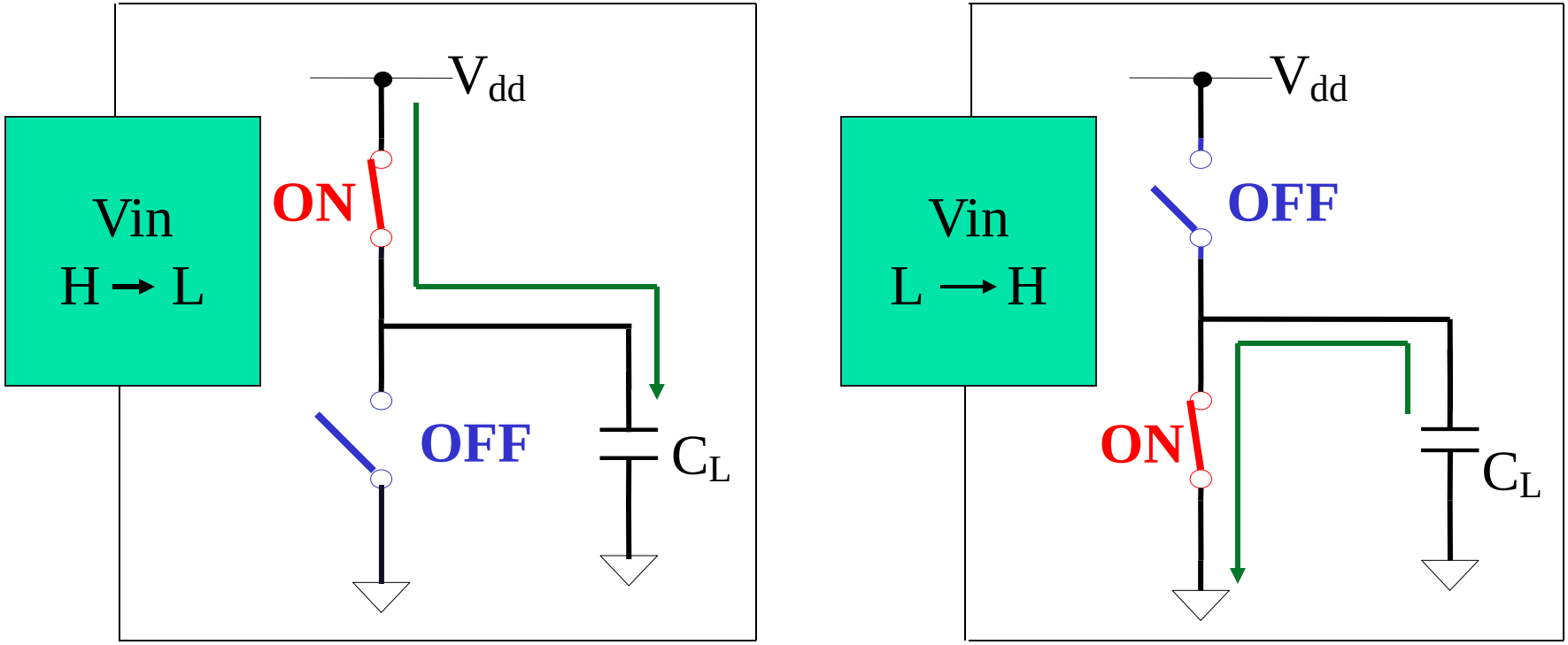
デジタルCMOS回路の静的電力消費は小さい



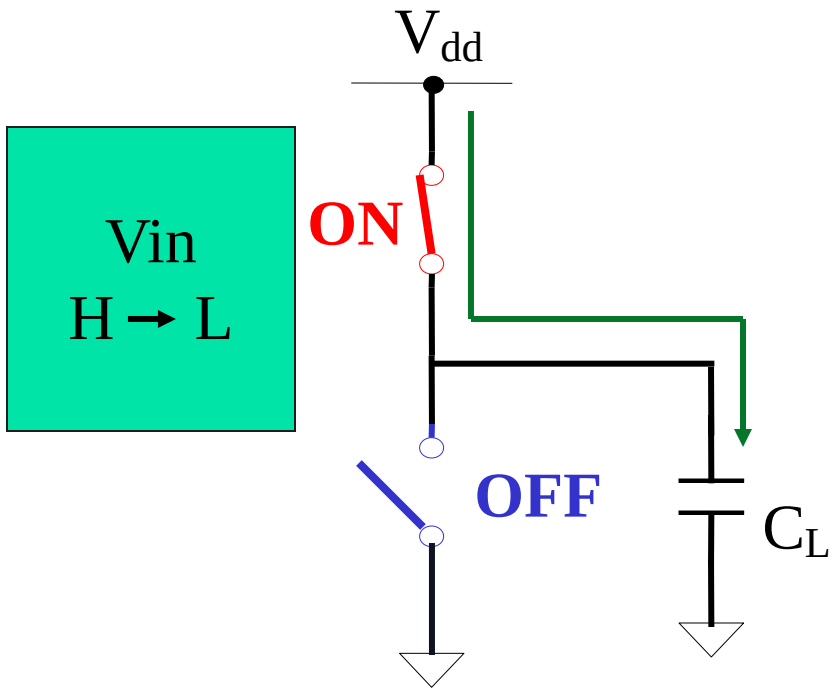


（注） 最近の微細CMOSデジタル回路では リーク電流が大きくなり、静的電力消費の占める割合が増えてきている。

デジタルCMOS回路の動的消費電力 (1)



デジタルCMOS回路の動的消費電力 (2)



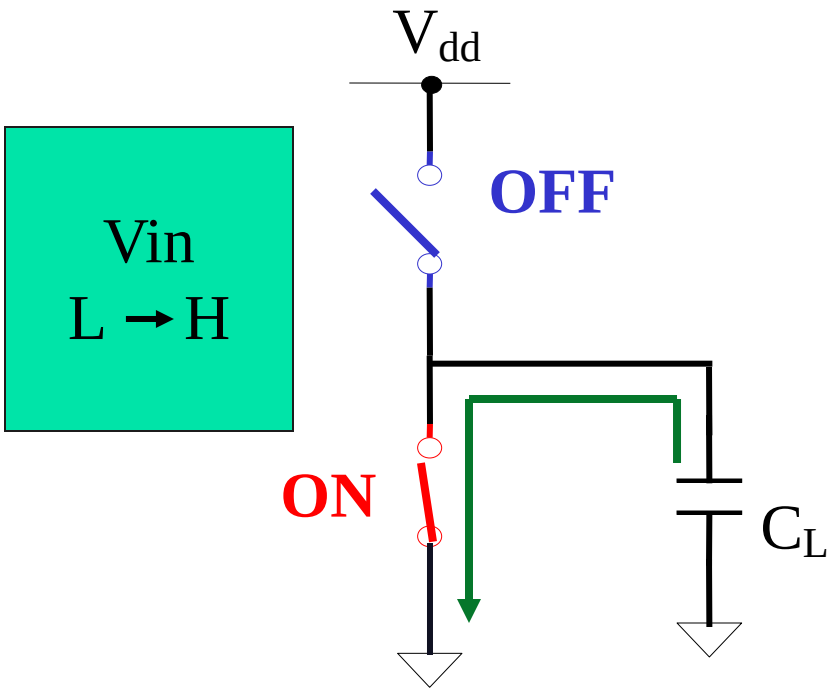
入力Vin：

High → Low

蓄積電荷Q：

0 → $C_L V_{dd}$

デジタルCMOS回路の動的消費電力 (3)



入力Vin：

Low → High

蓄積電荷Q：

$C_L V_{dd}$ → 0

デジタルCMOS回路の動的消費電力 (4)

Vout :H ⟶ L ⟶ H のとき

電荷 Q＝$C_L V_{dd}$ が電源 V_{dd}から GND へ流れる。

一秒間に出力が f 回のトグルするとき

Vdd からGNDへ流れるトータルの電荷 $Q_{total} = f\, C_L\, V_{dd}$

∴ 消費電力

$$P = V_{dd} \cdot I$$
$$= V_{dd}(f \cdot C_L \cdot V_{dd})$$
$$= f \cdot C_L \cdot V_{dd}^{\,2}$$

f :出力トグル周波数　C_L :負荷容量

V_{dd} :電源電圧

デジタルCMOS VLSIの低消費電力化

低消費電力化は大きな技術的課題

例： 携帯電話 ⇨ バッテリーが長持ちさせる

低消費電力化技術 ⇨ f, C_L, Vdd を小さくする。

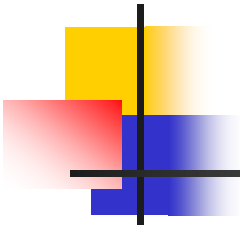
技術のトレンド：

周波数 f ：マイクロプロセッサのクロック周波数はより高くなる。 x

寄生容量C_L： 半導体の微細化により寄生容量は小。 ○

電源電圧Vdd: より低くして用いる。

5V ⇨ 3.3V ⇨ 1.8V ⇨ 1V ○



まとめ

- DSPは今後ますます重要な技術。
- DSPシステムは　DSPチップと
  アナログとのインターフェースの回路から
  構成される。
- 幅広いエレクトロニクス技術開発には
  デジタル、アナログ　両方の知識・技術が
  必要。