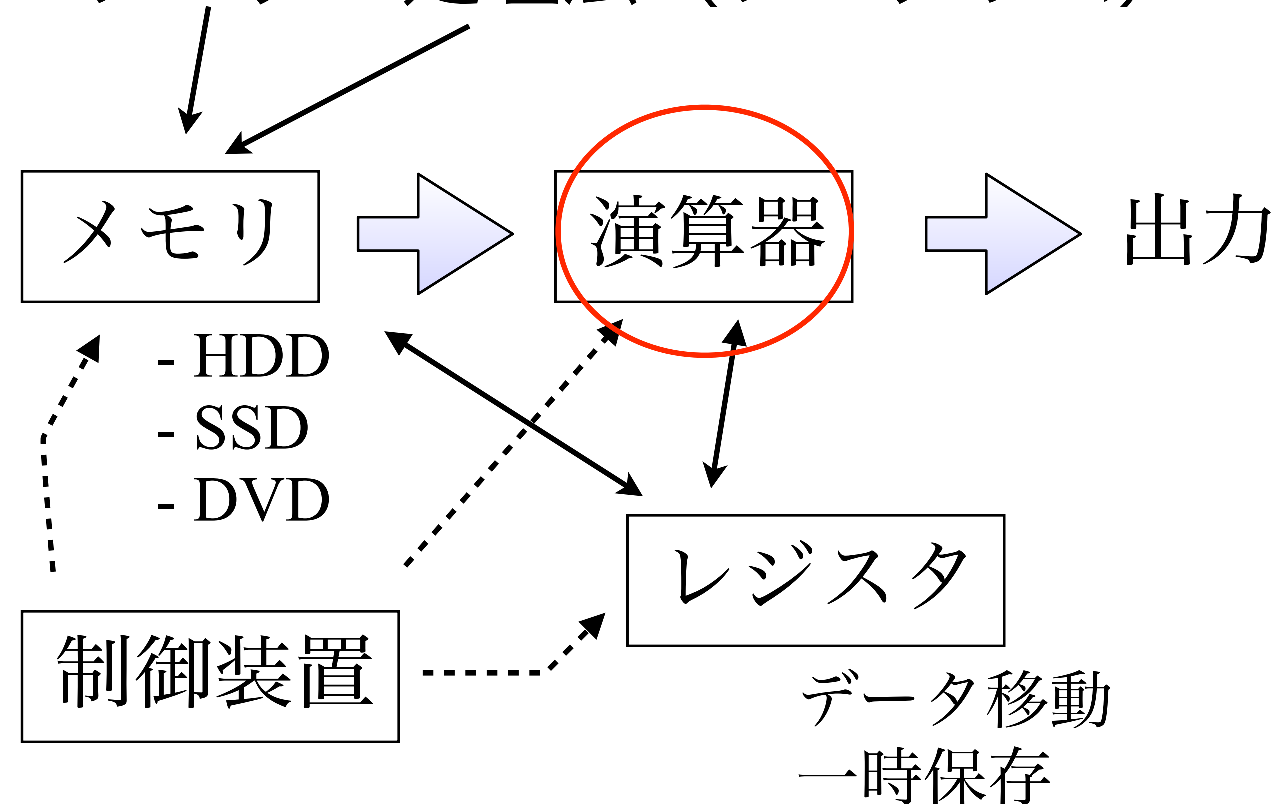
モデルコンピュータ（復習）

- ソフトウェア＝データ＋処理法（プログラム）



この４つの要素を適切につないだもの＝コンピュータ
つなぎ方や装置の構造構成（**アーキテクチャ**）にはいろいろある

今日は演算器をやります

演算器：論理演算を行う部品

加算器：足し算をする部品

a	b	c	c’	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

a	b	c’	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

桁上がり無し

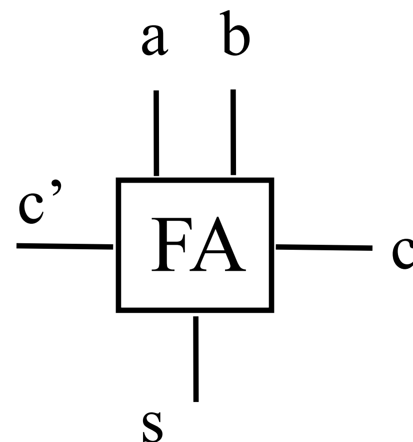
=半加算器（Half Adder)

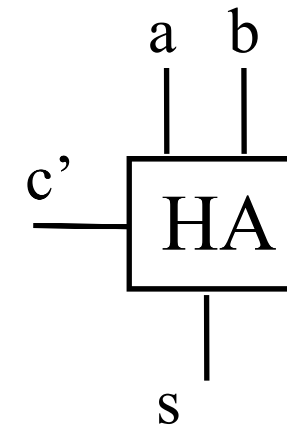
＝**HA**

桁上がり有り

＝全加算器（Full Adder)

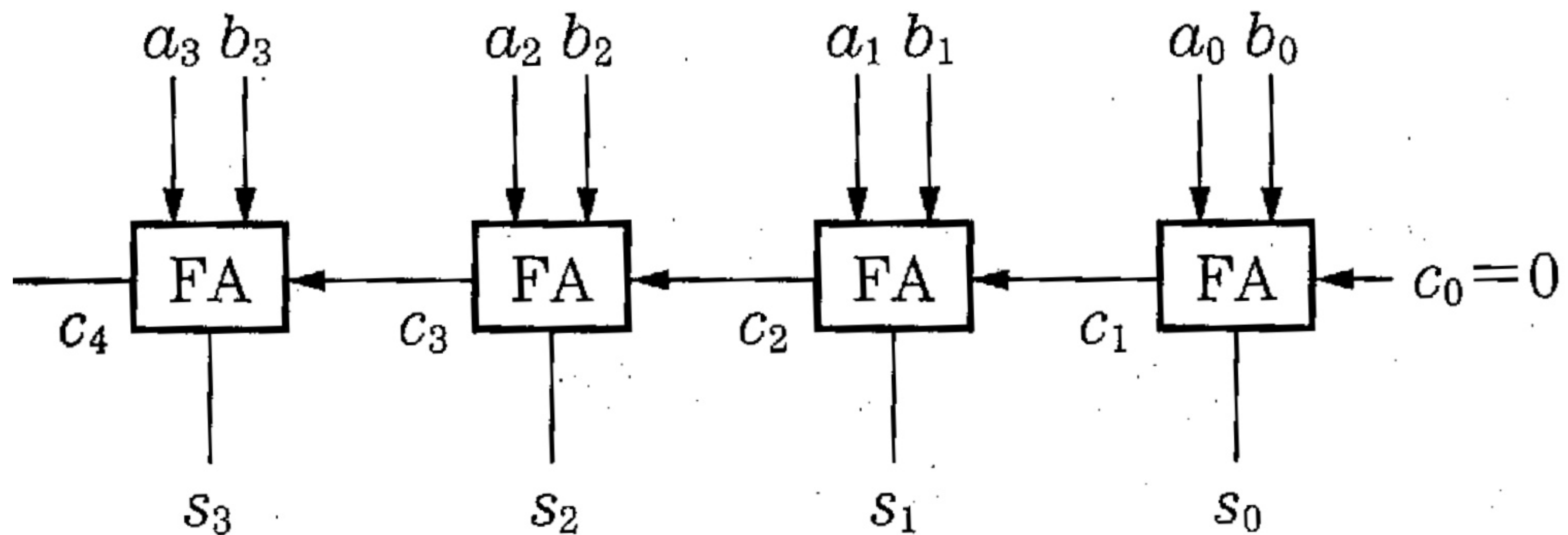
=**FA**





並列加算器：複数桁の足し算

- 4 bit加算器
- 結果が5桁になると桁溢れ（overflow)を起こす（計算エラー）



桁数を増やしたければ、FAを増やせば良い

しかし、桁上がり（c）隣のFAに伝えるのに時間がかかる

桁上げ先見加算器

桁上がり伝搬時間を減らす工夫を考えるためにＦＡを見直す

a	b	c	c'	s
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

桁上がり（キャリー）が0から1になる場合
a = b = 1 (1+1で桁が上がる)

$$g = ab$$

generation (創成)

OR

キャリーそのまま伝搬する場合

$$p = a \oplus b$$

propagation（伝搬）

OR

その他：キャリーは無し

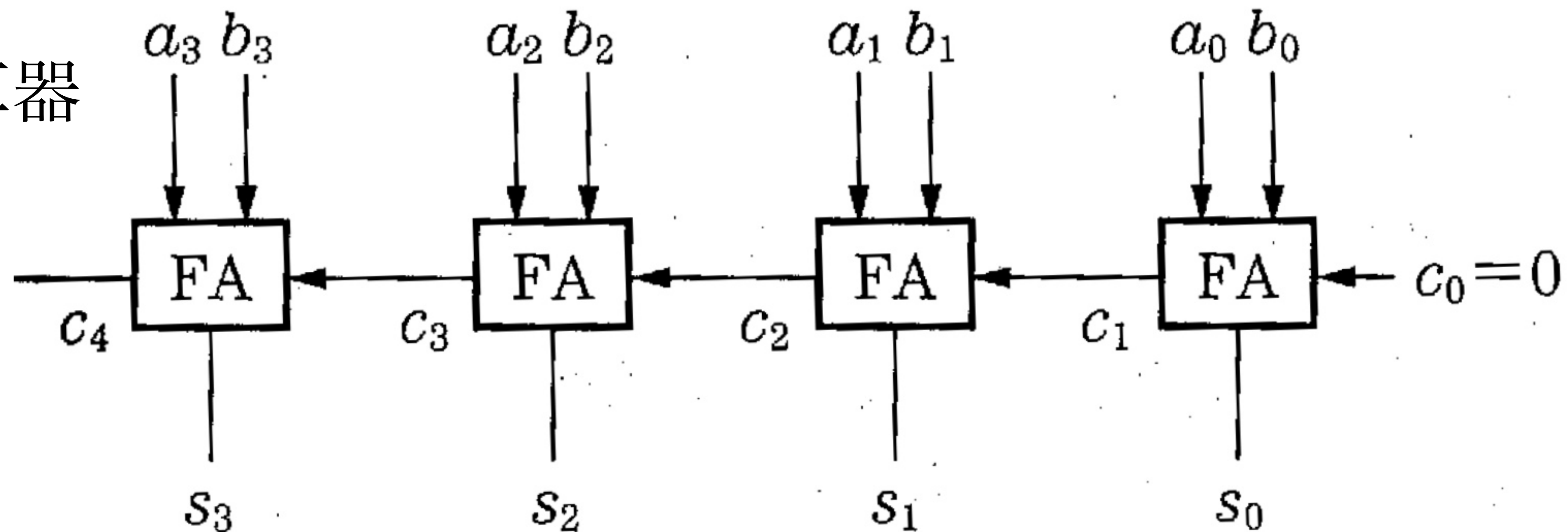
0

$$c' = ab + (a \oplus b)c = g + pc$$

ちゃんとした証明は真理値表で（省略）

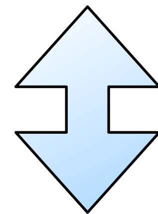
桁上げ先見加算器

並列加算器



桁数を増やしたければ、FAを増やせば良い

しかし、桁上がり（c）隣のFAに伝えるのに時間がかかる

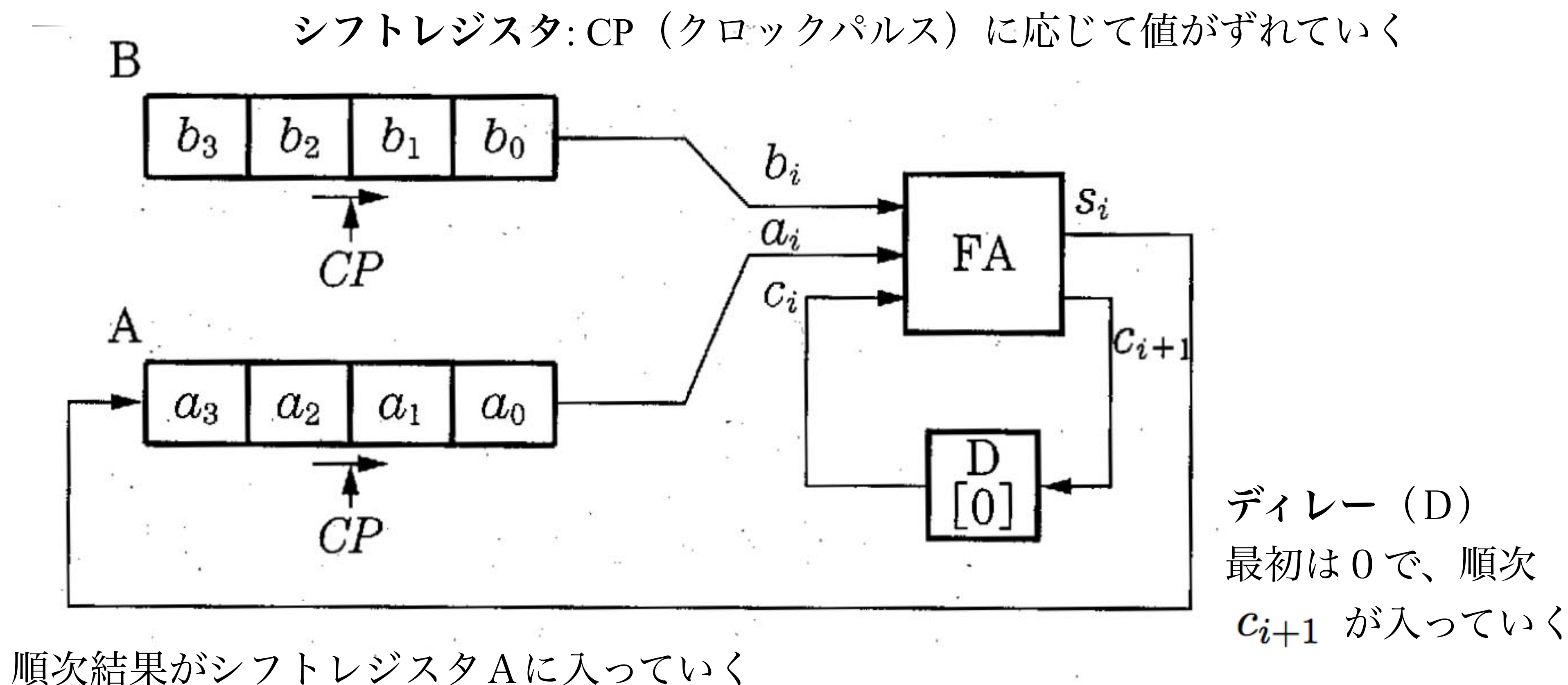


$c_{i+1} = a_i b_i + (a_i \oplus b_i) c_i$ =繰り返してciを消す $= \sum_{j=0}^{i} a_j b_j \prod_{k=j+1}^{i} (a_k \oplus b_k)$

a_k, b_k から直接 c_{i+1} を求める事が出来るので、伝搬時間が不要になる！

伝搬時間を気にせずＦＡを増やせる

直列加算器：１つのFAでn桁の足し算

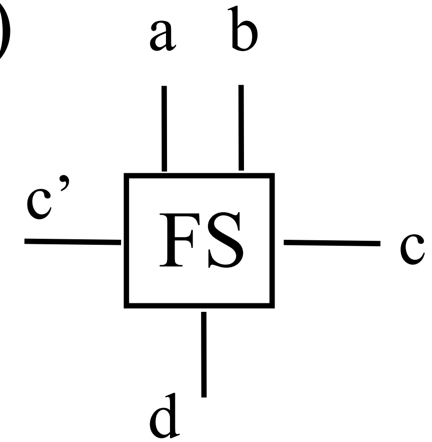


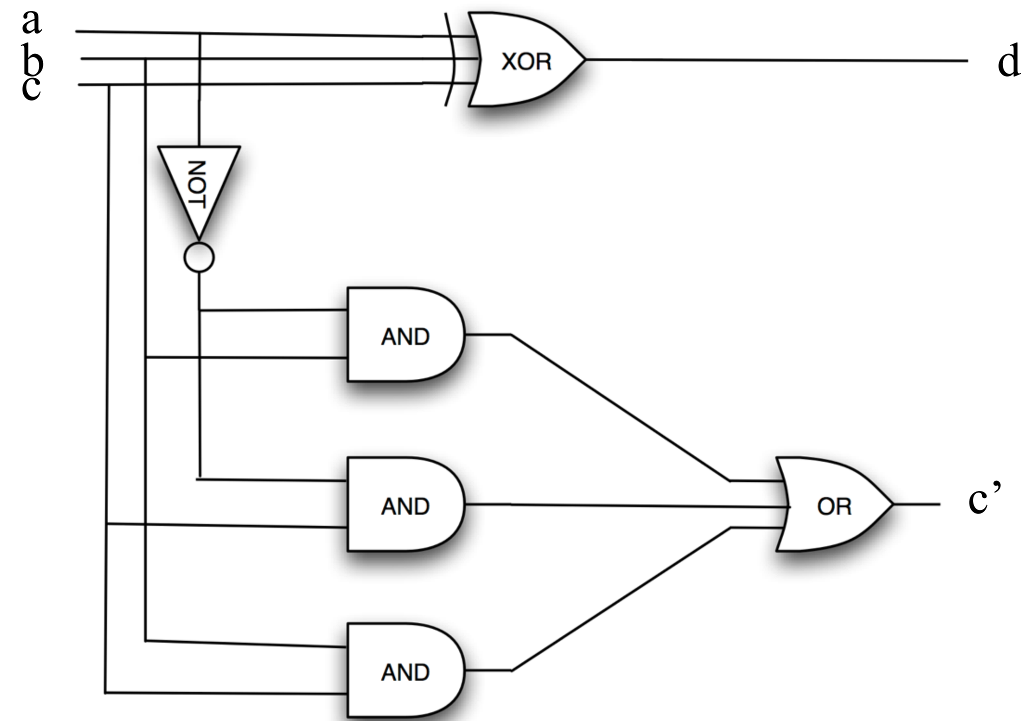
- スピードを考えると並列加算器の方が有利
- 回路面積、消費電力量などでは直列が有利
- 必要に応じて回路を使い分ける必要がある

減算器

全減算器FS (Full Subtractor)

a	b	c	c'	d
0	0	0	0	0
0	1	0	1	1
1	0	0	0	1
1	1	0	0	0
0	0	1	1	1
0	1	1	1	0
1	0	1	0	0
1	1	1	1	1





並列加算器や直列加算器とほぼ同様にn桁の減算器も作れる

負の数の表現

- いろいろ考え得る
  - 符号１ビット＋ｎビットでｎ桁の数を表現
  - 2進数の0,1を反転させたものを負の数とする（1の補数表現）
  - ０を決めて、その数との大小で決める（ゲタバキ表現）
- 加減算が便利になるように決める（2の補数表現）
  - a + b = 0となるbを-aだと思う

3bitでの例:

8(10) = [1000]は下位3bitを見ると０なので、

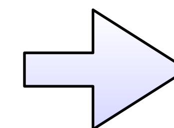
3 に対して、5を-3だと思うことにすると

$3 + 5 = 8 \rightarrow 0$

と見なせる

一般的には $[a_{m-1} \cdots a_1 a_0]^s = -a_{m-1} 2^{m-1} + \sum_{i=0}^{m-2} a_i 2^i$

ちなみに2進数の一般式：$[a_{n-1} \cdots a_1 a_0]_2 = \sum_{i=0}^{n-1} a_i 2^i$



m=3での例

10進数	2進数	符号付
0	[000]	0
1	[001]	1
2	[010]	2
3	[011]	3
4	[100]	-4
5	[101]	-3
6	[110]	-2
7	[111]	-1

1桁目を符号bitとも見なせる

2の補数表現（続き）

- ２の補数表現を採用すると、一般にm桁の符号付き2進数は
  - 正数： $1 \leq x \leq 2^{m-1} - 1$
  - 負数： $-1 \geq x \geq -2^{m-1}$

  の値の範囲を取る

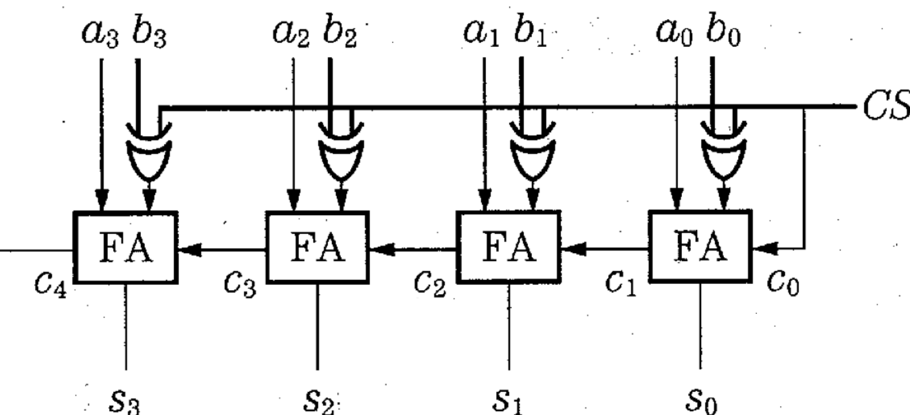
m=3での例

10進数	2進数	符号付
0	[000]	0
1	[001]	1
2	[010]	2
3	[011]	3
4	[100]	-4
5	[101]	-3
6	[110]	-2
7	[111]	-1

- ２の補数表現（正負反転）の簡単な求め方
  - 右の例から
    - $[001]^s = [111]$
    - $[010]^s = [110]$
  - 結局「**0, 1を反転して1を加える**」だけ
    - [001] →[110]→[111]
    - [010] →[101]→[110]
- ２の補数表現を用いると加減算器を構成できる

$$v = a + b = a + (+b) = a + b + 0$$
$$v = a - b = a + (-b) = a + [\overline{< b >}] + 1$$

反転は $\overline{b_i} = b_i \oplus 1$, 反転しないのは $b_i = b_i \oplus 0$ で実現



乗算器：正の２進数の掛け算

- 基本：

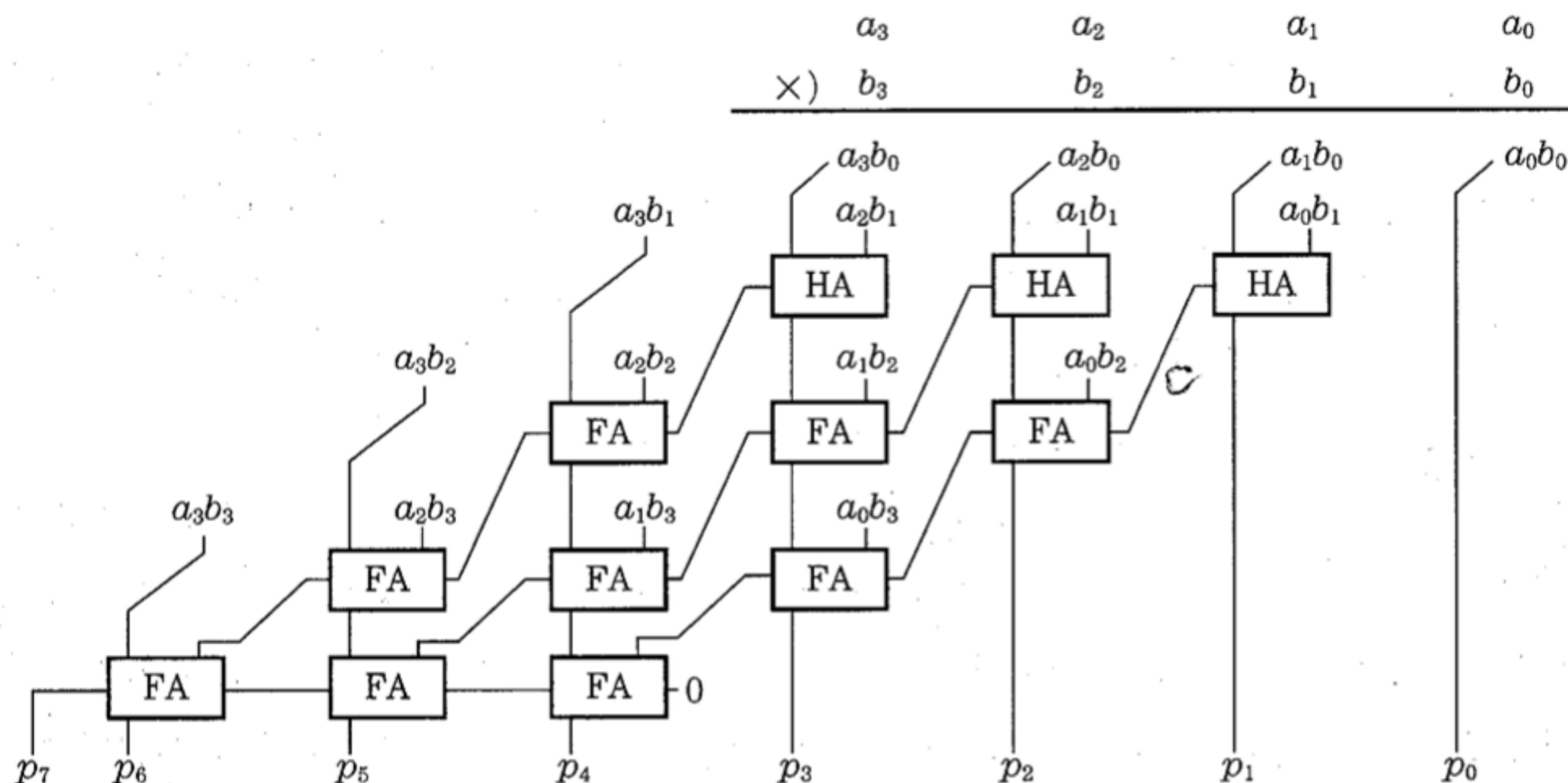
$$\begin{array}{r}0\\ \times\ 0\\ \hline 0\end{array}\quad \begin{array}{r}1\\ \times\ 0\\ \hline 0\end{array}\quad \begin{array}{r}0\\ \times\ 1\\ \hline 0\end{array}\quad \begin{array}{r}1\\ \times\ 1\\ \hline 1\end{array}$$

- 2桁以上

$$\begin{array}{rl}101 & \ldots 5\\ \times\ \ 10 & \ldots 2\\ \hline 000 & \\ 101\ \ & \\ \hline 1010 & \ldots 10\end{array}$$

- 4桁で一般化
  - n 桁 x n 桁＝2n 桁

$$\begin{array}{cccccccc} & & & & a_3 & a_2 & a_1 & a_0\\ & & & \times & b_3 & b_2 & b_1 & b_0\\ \hline & & & & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0\\ & & & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & \\ & & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & \\ & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & \\ \hline P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0\end{array}$$



筆算を桁上がりに注意して回路にする

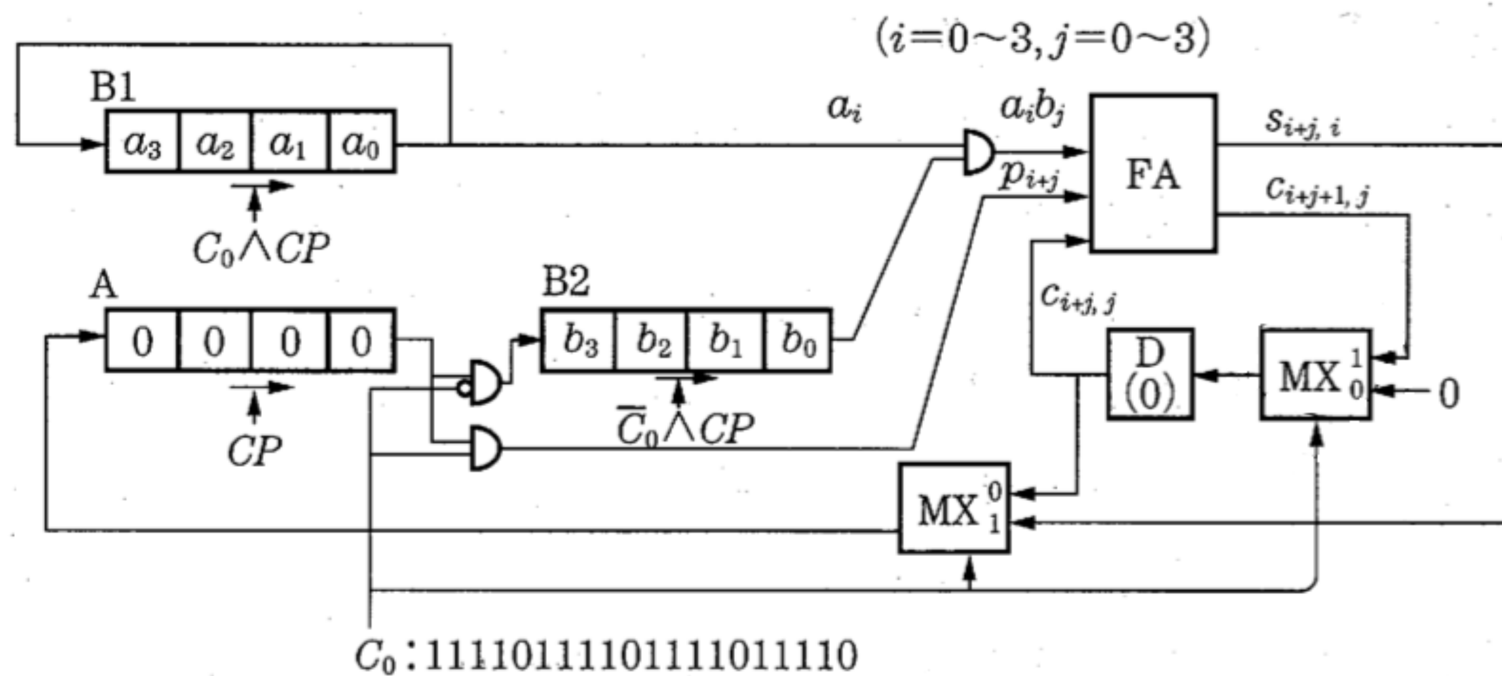
負の数は少しややこしくなる

直列乗算器

				a_3	a_2	a_1	a_0
			$\times$	b_3	b_2	b_1	b_0
				a_3b_0	a_2b_0	a_1b_0	a_0b_0
			a_3b_1	a_2b_1	a_1b_1	a_0b_1	
		a_3b_2	a_2b_2	a_1b_2	a_0b_2		
	a_3b_3	a_2b_3	a_1b_3	a_0b_3			
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

←bをシフトしながらANDを取って和を取る

シフトとFAで実現できる（直列乗算器）



制御信号（左から見る）

結果はシフトレジスタA, B2に入る

MX: マルチプレクサ

2^n 本の入力線からn個の制御信号で１つを選ぶ（次ページ）

マルチプレクサ（制御器）

マルチプレクサ

2^n 本の入力線からn個の制御信号で１つを選ぶ

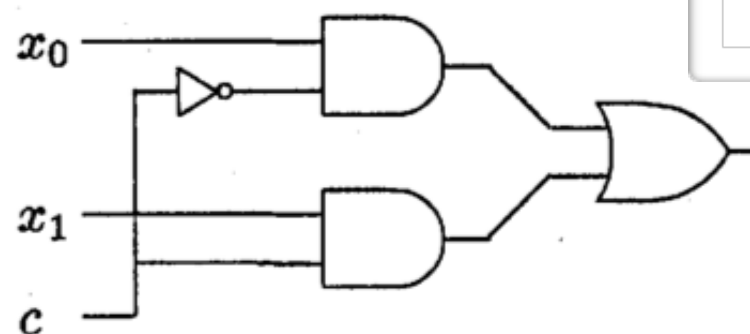
n=1の場合

２本から１個の制御信号で１つを選ぶ

x0	x1	c	*MX1*
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

c = 0 → x0
c = 1 → x1

$$MX_1 = x_0\bar{c} + x_1 c$$



n=2の場合

4本から2個の制御信号で2つを選ぶ

c0	c1	*MX1*
0	0	x0
0	1	x1
1	0	x2
1	1	x3

$$MX_2 = x_0\bar{c}_0\bar{c}_1 + x_1\bar{c}_0 x_1 + x_2 c_0\bar{c}_1 + x_3 c_0 c_1$$

$$MX_n = \sum_{(e_1 e_2 \cdots e_n) \in B^n} x_{[e_1 \cdots e_n]} c_1^{e_1} \cdots c_n^{e_n}$$

一般的にnの場合

デマルチプレクサ

- マルチプレクサの逆

マルチプレクサ

2^n 本の入力線からn個の制御信号で１つを選ぶ

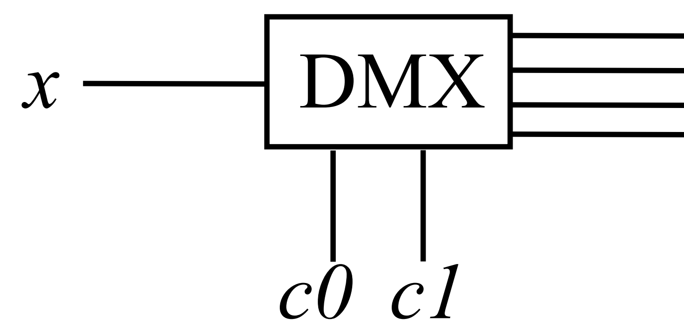
$$MX_n = \sum_{(e_1 e_2 \cdots e_n) \in B^n} x_{[e_1 \cdots e_n]} c_1^{e_1} \cdots c_n^{e_n}$$

デマルチプレクサ

n個の制御信号で１つの入力を 2^n へ分ける

$$DMX_n = x c_1^{e_1} \cdots c_n^{e_n}$$

n = 2の例



$$y_0 = x c_0^0 c_1^0 = x \bar{c}_0 \bar{c}_1$$

$$y_1 = x c_0^0 c_1^1 = x \bar{c}_0 c_1$$

$$y_2 = x c_1^1 c_1^0 = x c_0 \bar{c}_1$$

$$y_3 = x c_1^1 c_1^1 = x c_0 c_1$$

比較器

- ２進数の大小：[00001101] > [00001001]
- 上位（左）ビットから見て、最初に異なるビットの大小で決まる

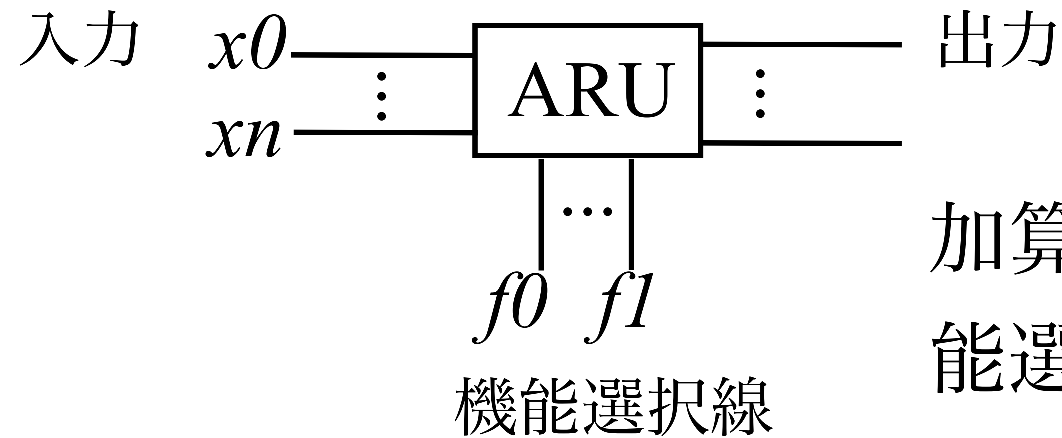
GT(a, b): aよりbが大きい事の判定器を考えよう

1. ビット列$(a_{n-1}\cdots a_1 a_0)$ $(b_{n-1}\cdots b_1 b_0)$ を考える
2. *i*番目のビットが一致している：$\overline{a_i \oplus b_i}$
3. k+1番目までの上位ビットは一致していた：$\prod_{i=k+1}^{n-1} \overline{a_i \oplus b_i}$
4. aよりbが大きい：$a_k < b_k \rightarrow a_k = 0, b_k = 1 \rightarrow \bar{a}_k b_k$
5. $k = 0 \sim n-1$の場合を考える。(3, 4のANDと全体のORを取る)

$$GT = \sum_{k=0}^{n-1} \bar{a}_k b_k \prod_{i=k+1}^{n-1} \overline{a_i \oplus b_i}$$

a, bを入れ替えるとLessが出来る。一致は単純に $Eq = \prod_{i=0}^{n-1} \overline{a_i \oplus b_i}$

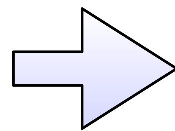
ALU（あるー）算術演算ユニット



加算、比較など異なる論理演算を機能選択線への入力で切り替える演算器

例：ANDとORの機能を持つALU

a	b	AND	OR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1



選択線の入力

a	b	f	X
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

AND（f = 0）

OR（f = 1）