H31 [2]

(1) 以下に疑似コードを示す。(なお、適宜 //(コメントアウト)により、その内容の補足を記している)

```
int popcount(int k) {

    int res = 0;
    for bit in {0 ··· 5}
        if {(k >> bit) & 1} res++;   // kの bit目が1かどうか判定

    return res;
}
```

まず $0 \le n \le 32$ より、高々 [0〜5] bit目まで判定すれば良い。kの bit番目の bitは (k>>bit) & 1 で得ることができ、これが1であれば、それは kの bit番目の bitが1である。これは1回のシフト演算と1回の論理演算で実現でき、そのコストは2単位時間である。
これを [0〜5] bit目まで順に行うことから、要する計算時間は $2 \times 6 = 12$ 単位時間 となる。

(2) あらかじめ(1)の方法等で、$0 < n \le 32$ に対する population count の値を計算しておき、表と 32 の配列に、順にその計算結果を前から保存しておくこととする。
すると、$0 < n \le 32$ に対する population count の値は、配列の前から n番目 (1-index) にアクセスすることで取得できる。これは、1回の配列アドレス計算と1回のアドレス参照で実現でき、そのコストは 0+1=1 単位時間である。以上より、表引き操作による計算時間は 1単位時間 となる。

(3) (1)と(2)の中間的な方式の疑似コードを示す。(なお、適宜コメントアウトにより、その内容を補足している。)

```
cache = [0] * 8    // cache[n] = (0≤n<8) に対する population_count の値
// cache[0..7]を計算
for n in {0 ··· 7}
    for bit in {0 ··· 3}
        if (n >> bit) & 1 : cache[n]++;

int popcount7(int n) {
    int res = 0;          // nの前3bitと後ろ3bitに分けて計算する。(この際、あらかじめ計算した
    res = cache[n & 7] + cache[ n >> 3 ];   cacheのデータを使用する)
    return res;
}
```

この方法は、まずあらかじめ $0 \le n \le 32$ ではなく、その1/4ほどの範囲の $0 \le n < 8$ に対して population_count の値を計算しておき、$0 \le n \le 32$ に対する population_count の際は 前3bitを表す (n&7) と、後ろ3bitを表す (n>>3) ごとに、前計算の結果より得て、その合計を返すという手法に基いている。

例えば $n = 27_{(10)} = \underbrace{011}_{②}\ \underbrace{011}_{①}{}_{(2)}$ に対して、n&7は①に対応し、n>>3&7は②に対応するという具合である。

cacheの計算コストを除けば、この疑似コードによるpopulation_countの計算コストは、1回の論理論算、1回のシフト演算、2回の表引き、1回の四則演算(加算)の合計とに解釈でき、その計算時間は4単位時間である。

これは(1)より高速であり、また要するストレージの量もその約 $\frac{1}{4}$ ほどである。

(4) 入力を低位の桁から順に I_0, I_1, I_2、出力のうち高位の方を S_1、低位の方を S_0 とする。

この時得られる真理値表は以下。

I_2	I_1	I_0	S_1	S_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

カルノー図を用いて S_1, S_0 の論理式を簡略化すると、

[S_1]

I_2 \ I_1I_0	00	01	11	10
0			1	
1		1	1	1

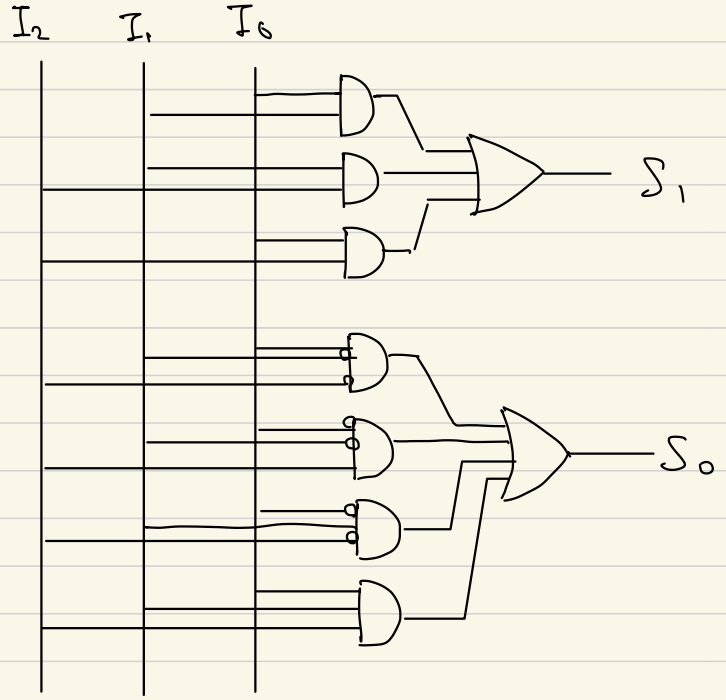
$S_1 = I_0I_1 + I_1I_2 + I_2I_0$

[S_0]

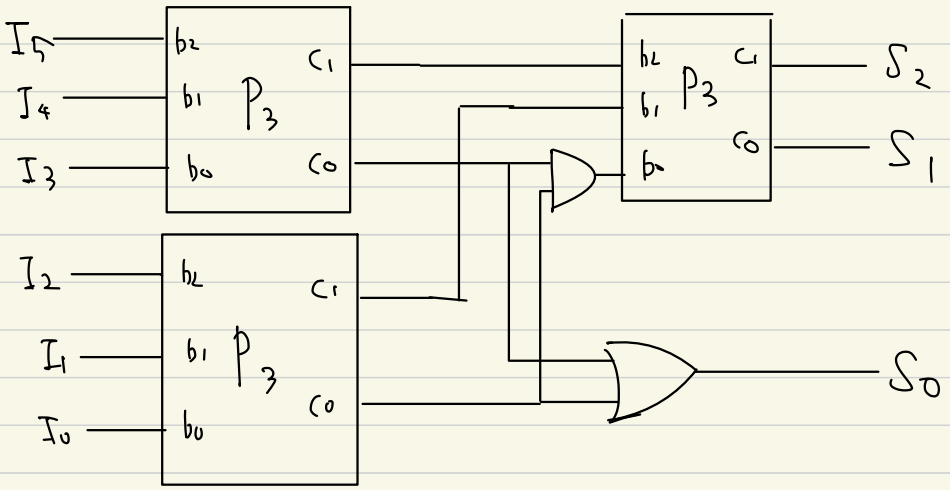
I_1 \ I_2I_0	00	01	11	10
0		1		1
1	1		1	

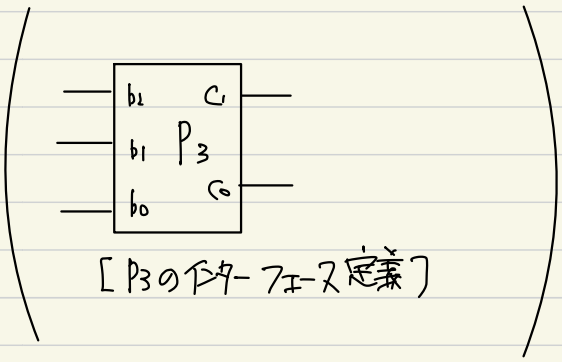
$S_0 = \overline{I_2}\,\overline{I_1}\,\overline{I_0} + \overline{I_2}\,I_1I_0 + I_2\,\overline{I_1}\,\overline{I_0} + I_2I_1I_0$

したがって、P_3 を設計すると以下のようになる。(表記法はMIL記法に基づく)



(5) 入力を低位から順に $I_0, I_1, I_2, I_3, I_4, I_5$. 出力を高位から順に S_2, S_1, S_0 とする。
この時. P_3 や AND, OR, NOT ゲートを用いて P_6 を作成すると以下のようになる.
(但し. P_3 のインターフェースは入力において高位から順に. b_2, b_1, b_0 で与えられ, 出力については高位から順に c_1, c_0 で与えられるものとする。右下図参照)





(6) 上のように, nビットをいくつかの連続bitグループごとに計算し, その総和をもとに計算するという方式では, その再帰の深さにたじて発生するゲート遅延の影響が相対的に大きくなる. これを回避するには, 再帰における小問題の出力を待つことなく, nビットの入力から直接その入力を計算する論理回路を設計することが挙げられる. これらは加算器の高速化においても用いられている技術であり「キャリールックアヘッド方式」と呼ばれる。設計に要する素子の個数がnに対して大きく増加するものの, その遅延コストは $O(n)$ から $O(\log n)$ 程度まで改善されることが期待できる.