

2번 문제 답안

벤치마크 결과는 **1261ms**이며, 로테이션 결과를 미리 table을 생성하고 짝수, 홀수 라운드를 통합하여 구현하였다.

1. 구현 및 속도 측정

속도 측정시 사용한 보드 모델은 UNO R3이며 아두이노 1.8.12 에서 구현하였다. 속도를 측정한 컴퓨터는 window 환경이며, 프로세서 Intel(R) Core(TM) i5-8265U CPU, RAM 16.0GB, 64비트 운영 체제입니다.

다음 표는 문제에서 주어진 코드에서 최적화를 위한 개선 사항과 그 개선 사항을 적용한 후 속도를 측정한 결과를 나타냅니다.

수정 함수명	순번	개선 사항	속도
-	1-1	없음 (주어진 코드 그대로 속도 측정)	3909ms
key_gen	2-1	ROL8(key_in[0], 1) + ROL8(key_in[0], 5) → Sbox1 ROL8(key_in[1], 3) + ROL8(key_in[1], 7) → Sbox2 값을 미리 계산하여 table로 저장	3826ms
	2-2	홀수, 짝수 라운드의 과정을 합쳐 if문을 제거하고 for문을 64번 반복하도록 수정	3559ms
	2-3	ROL16(con, i%16) 값을 table로 만들어 미리 저장	2009ms
	2-4	ROL16(*key_p, 1)+ROL16(*key_p, 9)를 배열을 사용하여 계산	1794ms
	2-5	for문으로 반복되는 연산을 모두 나열하여 for문 제거	1650ms
enc	3-1	홀수, 짝수 라운드의 과정을 합쳐 if문을 제거하고 for문을 64번 반복하도록 수정	1304ms
	3-2	for문으로 반복되는 연산을 모두 나열하여 for문 제거	1261ms
		위 개선 사항 모두 적용	1261ms

[표 1] 개선 사항

2번 문제 답안

2. 개선 사항 세부 설명

[표 1]에서 제시한 순번 순으로 개선한 방법은 아래와 같다.

<key_gen>

순번 2-1. $\text{ROL8}(\text{key_in}[0], 1) + \text{ROL8}(\text{key_in}[0], 5)$ 와 $\text{ROL8}(\text{key_in}[1], 3) + \text{ROL8}(\text{key_in}[1], 7)$ 에서 $\text{key_in}[0]$ 과 $\text{key_in}[1]$ 이 될 수 있는 값은 0~255이다. 각 값에 대해 결과 값이 결정적이기 때문에 미리 계산하여 table로 저장한다. 저장한 table의 이름은 각 Sbox1, Sbox2이다.

```
const u8 Sbox1[256] = {0, 34, 68, 102, 136, 170, 204, 238, 17, 51, 85, 119, 153, 187, 221, 255,
34, 68, 102, 136, 170, 204, 238, 16, 51, 85, 119, 153, 187, 221, 255, 33,
68, 102, 136, 170, 204, 238, 16, 50, 85, 119, 153, 187, 221, 255, 33, 67,
102, 136, 170, 204, 238, 16, 50, 84, 119, 153, 187, 221, 255, 33, 67, 101,
136, 170, 204, 238, 16, 50, 84, 118, 153, 187, 221, 255, 33, 67, 101, 135,
170, 204, 238, 16, 50, 84, 118, 152, 187, 221, 255, 33, 67, 101, 135, 169,
204, 238, 16, 50, 84, 118, 152, 186, 221, 255, 33, 67, 101, 135, 169, 203,
238, 16, 50, 84, 118, 152, 186, 220, 255, 33, 67, 101, 135, 169, 203, 237,
17, 51, 85, 119, 153, 187, 221, 255, 34, 68, 102, 136, 170, 204, 238, 16,
51, 85, 119, 153, 187, 221, 255, 33, 68, 102, 136, 170, 204, 238, 16, 50,
85, 119, 153, 187, 221, 255, 33, 67, 102, 136, 170, 204, 238, 16, 50, 84,
119, 153, 187, 221, 255, 33, 67, 101, 136, 170, 204, 238, 16, 50, 84, 118,
153, 187, 221, 255, 33, 67, 101, 135, 170, 204, 238, 16, 50, 84, 118, 152,
187, 221, 255, 33, 67, 101, 135, 169, 204, 238, 16, 50, 84, 118, 152, 186,
221, 255, 33, 67, 101, 135, 169, 203, 238, 16, 50, 84, 118, 152, 186, 220,
255, 33, 67, 101, 135, 169, 203, 237, 16, 50, 84, 118, 152, 186, 220, 254};

const u8 Sbox2[256] = {0, 136, 17, 153, 34, 170, 51, 187, 68, 204, 85, 221, 102, 238, 119, 255,
136, 16, 153, 33, 170, 50, 187, 67, 204, 84, 221, 101, 238, 118, 255, 135,
17, 153, 34, 170, 51, 187, 68, 204, 85, 221, 102, 238, 119, 255, 136, 16,
153, 33, 170, 50, 187, 67, 204, 84, 221, 101, 238, 118, 255, 135, 16, 152,
34, 170, 51, 187, 68, 204, 85, 221, 102, 238, 119, 255, 136, 16, 153, 33,
170, 50, 187, 67, 204, 84, 221, 101, 238, 118, 255, 135, 16, 152, 33, 169,
51, 187, 68, 204, 85, 221, 102, 238, 119, 255, 136, 16, 153, 33, 170, 50,
187, 67, 204, 84, 221, 101, 238, 118, 255, 135, 16, 152, 33, 169, 50, 186,
68, 204, 85, 221, 102, 238, 119, 255, 136, 16, 153, 33, 170, 50, 187, 67,
204, 84, 221, 101, 238, 118, 255, 135, 16, 152, 33, 169, 50, 186, 67, 203,
85, 221, 102, 238, 119, 255, 136, 16, 153, 33, 170, 50, 187, 67, 204, 84,
221, 101, 238, 118, 255, 135, 16, 152, 33, 169, 50, 186, 67, 203, 84, 220,
102, 238, 119, 255, 136, 16, 153, 33, 170, 50, 187, 67, 204, 84, 221, 101,
238, 118, 255, 135, 16, 152, 33, 169, 50, 186, 67, 203, 84, 220, 101, 237,
119, 255, 136, 16, 153, 33, 170, 50, 187, 67, 204, 84, 221, 101, 238, 118,
255, 135, 16, 152, 33, 169, 50, 186, 67, 203, 84, 220, 101, 237, 118, 254};
```

수정 전

```
for (i = 0; i < ROUND_NUM; i++) {
    if (i % 2 == 0) {
        key_in[0] = ROL8(key_in[0], 1) + ROL8(key_in[0], 5);
        key_in[1] = ROL8(key_in[1], 3) + ROL8(key_in[1], 7);
    } else {
        *key_p = ROL16(*key_p, 1) + ROL16(*key_p, 9) + ROL16(con, (i%16));
    }
}
```

수정 후

```
for (i = 0; i < ROUND_NUM; i++) {
    if (i % 2 == 0) {
        key_in[0] = Sbox1[key_in[0]];
        key_in[1] = Sbox2[key_in[1]];
    } else {
        *key_p = ROL16(*key_p, 1) + ROL16(*key_p, 9) + ROL16(con, (i%16));
    }
}
```

2번 문제 답안

순번 2-2. 홀수, 짝수 라운드의 연산 과정이 달라 기존 코드는 이를 if문으로 구분한다. 따라서 홀수, 짝수 라운드를 하나로 묶고 if문을 제거해 for문을 64번 반복하도록 수정한다. 이때, for문의 i를 2씩 증가로 변경하고, i에 맞추어 연산을 몇 가지 수정하였다 (빨간색으로 표시).

```
int i;
for (i = 0; i < ROUND_NUM; i+=2) {

    key_in[0] = Sbox1[key_in[0]];
    key_in[1] = Sbox2[key_in[1]];

    tmp1 = key_in[0] + key_in[1];
    tmp2 = key_in[0] ^ key_in[1];

    key_in[0] = tmp1;
    key_in[1] = tmp2;
    rnd[i * 2 + 0] = key_in[0];
    rnd[i * 2 + 1] = key_in[1];

    *key_p = ROL16(*key_p, 1) + ROL16(*key_p, 9) + ROL16(con, (i+1)%16);

    tmp1 = key_in[0] + key_in[1];
    tmp2 = key_in[0] ^ key_in[1];

    key_in[0] = tmp1;
    key_in[1] = tmp2;
    rnd[i * 2 + 2] = key_in[0];
    rnd[i * 2 + 3] = key_in[1];
}
```

순번 2-3. 기존 코드에서 ROL16(con, i%16)은 i가 홀수인 경우에만 사용된다. i가 홀수일 때, i%16이 될 수 있는 값은 1, 3, 5, 7, 9, 11, 13, 15로 8개이고, 각 값이 반복되어 적용된다. 그리고 8개의 값은 결정적이다. 따라서 미리 계산하여 table로 저장하여 사용한다.

```
const ul6 con[8] = {0x3579, 0xD5E4, 0x5793, 0x5E4D, 0x7935, 0xE4D5, 0x9357, 0x4D5E};
```

i가 0에서 127까지 2씩 증가 할 때, 배열 con의 인덱스가 0부터 7까지 반복되게 하기 위해서 (i>>1)&7을 사용했다.

i	(i>>1)&7	i	(i>>1)&7
0	0	16	0
2	1	18	1
4	2	20	2
6	3	22	3
8	4	24	4
10	5	26	5
12	6	28	6
14	7	30	7

2번 문제 답안

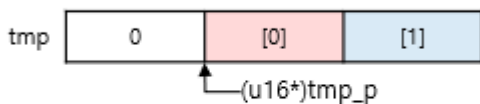
순번 2-4. $\text{ROL16}(*\text{key_p}, 1)$ 와 $\text{ROL16}(*\text{key_p}, 9)$ 의 값은 앞 8비트와 뒤 8비트를 swap한 것과 같다.



이 사실을 기반으로 로테이션을 2번 하지 않고 배열을 사용하는 방법을 사용해 $\text{ROL16}(*\text{key_p}, 1) + \text{ROL16}(*\text{key_p}, 9)$ 을 계산하도록 수정하였다.



1) 크기 3인 배열 tmp를 생성한다.



2) tmp[1], tmp[2] 위치에 *key_p<<1 값을 저장하고, tmp_p가 tmp[1]의 주소를 가르키게 선언한다.



3) tmp[2]의 값을 tmp[0]에 복사하고, key_p가 tmp[0]의 주소를 가르키도록 선언한다.

4) *tmp_p와 *key_p를 더한다.

tmp_p가 가리키는 값은 *key_p<<1이고 key_p가 가리키는 값은 *key_p<<9인 형태가 된다.

따라서 *tmp_p + *key_p = $\text{ROL16}(*\text{key_p}, 1) + \text{ROL16}(*\text{key_p}, 9)$ 이 된다.

순번 2-5. for문에서 수행되는 연산을 모두 줄이기 위해 for문 안 연산을 모두 풀어 for문을 제거한다.

```
key_in[0] = Sbox1[key_in[0]];
key_in[1] = Sbox2[key_in[1]];

tmp1 = key_in[0] + key_in[1];
tmp2 = key_in[0] ^ key_in[1];
key_in[0] = tmp1;
key_in[1] = tmp2;
rnd[0] = key_in[0];
rnd[1] = key_in[1];

*(u16*)(tmp+1) = ROL16(*key_p, 1);
tmp[0] = tmp[2];
*key_p = *(u16*)(tmp+1) + *tmp_p + con[0];

tmp1 = key_in[0] + key_in[1];
tmp2 = key_in[0] ^ key_in[1];
key_in[0] = tmp1;
key_in[1] = tmp2;
rnd[2] = key_in[0];
rnd[3] = key_in[1];

key_in[0] = Sbox1[key_in[0]];
key_in[1] = Sbox2[key_in[1]];
```

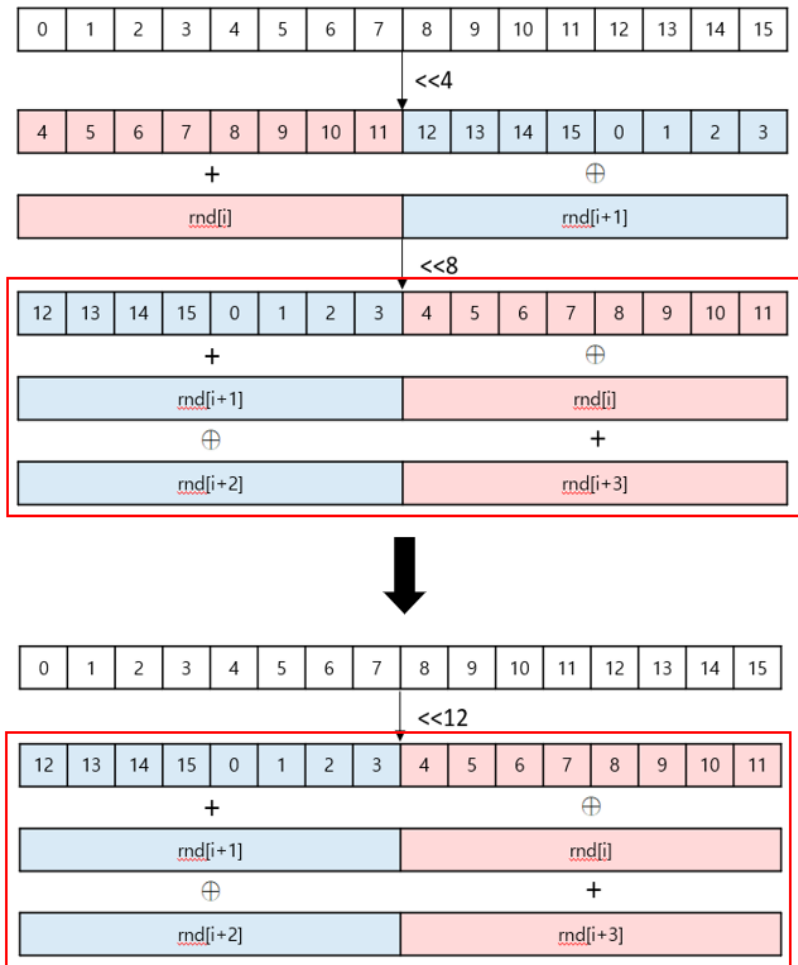
[코드의 일부]

2번 문제 답안

<enc>

순번 3-1. 홀수, 짝수 라운드 연산 과정이 달라 if문으로 구분한다. 홀수, 짝수 라운드 과정을 하나로 합쳐 if문을 제거한다.

기존 연산에서는 짝수 라운드에서 $\ll 4$, 홀수 라운드에서 $\ll 8$ 를 하지만 라운드 키 연산 위치를 바꿔주면 한번에 $\ll 12$ 를 해도 같은 결과가 나온다.




순번 3-2. for문에서 수행되는 연산을 모두 줄이기 위해 for문 안 연산을 모두 풀어 for문을 제거한다.

2번 문제 답안

3. 테스트 벡터 확인 결과 및 벤치마크 결과

위 개선 사항을 모두 적용해 코드를 수정 후, 실행하면 다음과 같이 결과가 나온다. 테스트 벡터를 모두 통과하였으며, 벤치마크 결과 1261ms이었다.

 COM3

```
|  
-----  
TEST VECTOR  
-----  
>> CORRECT  
>> CORRECT  
>> CORRECT  
-----  
BENCHMARK  
-----  
>> 1261  
-----
```