AES & Inline assembly

한성대 김경호

https://www.youtube.com/watch?v=gj4CHqilWvw

목차

1. AES 란?

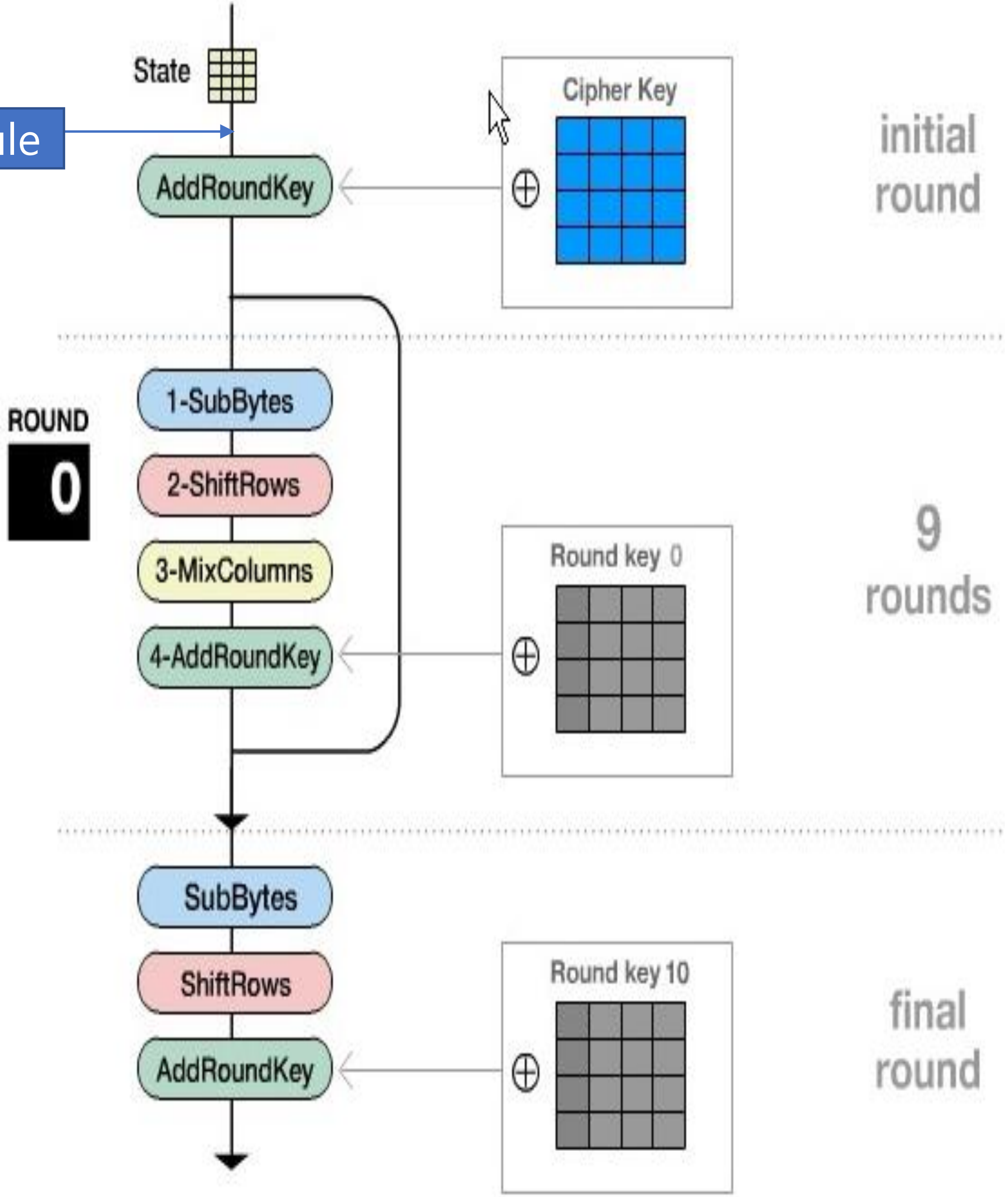
2. 기본적인 AVR assembly

3. C & inline assembly를 이용한 AES 구현

4. Atmel Studio를 이용한 디버깅

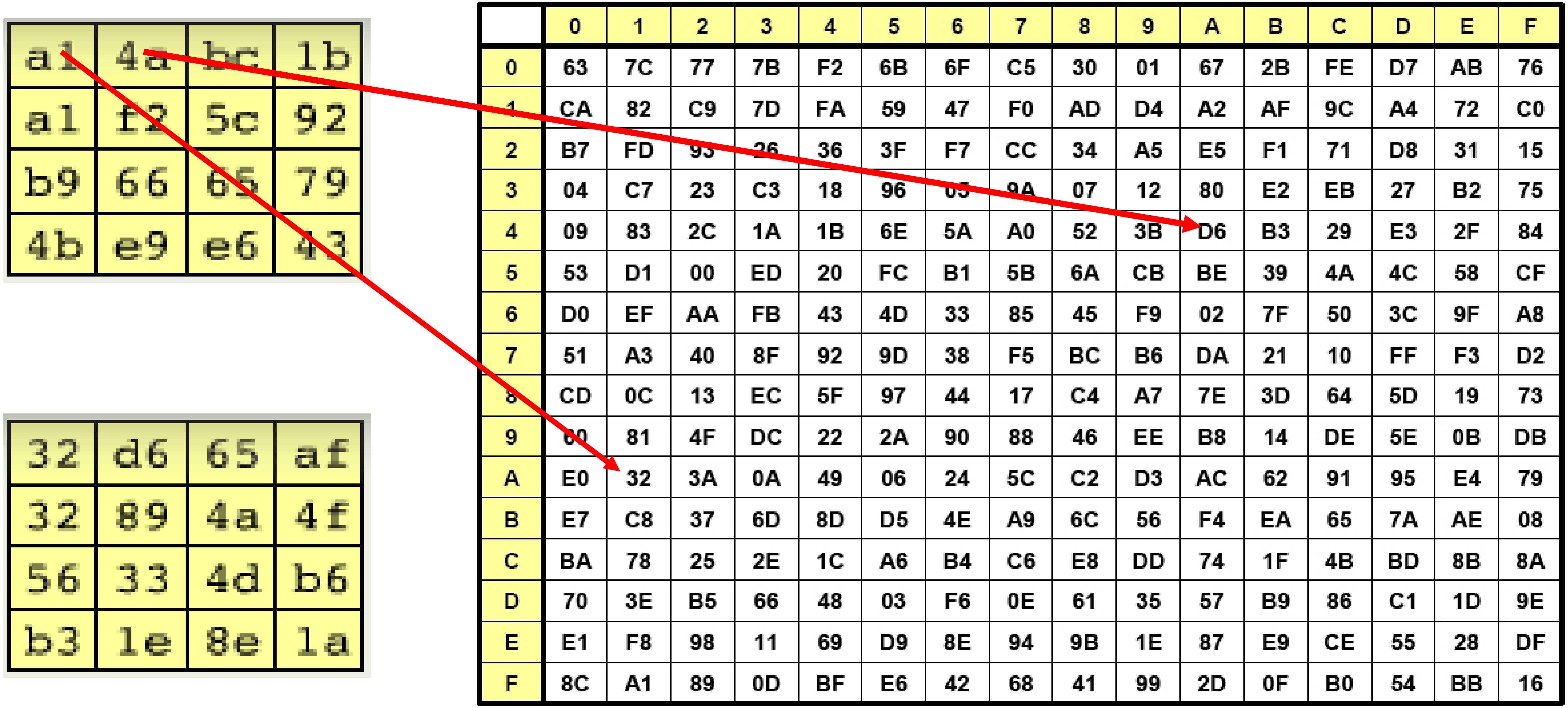
1. AES 란?

- 현재 가장 많이 쓰이는 대칭형 암호화 알고리즘
- 뛰어난 안정성과 속도
- AES-128, AES-192, AES-256
- SubBytes, ShiftRows, MixColumns, AddRoundKey 연산을 반복



KeySchedule

SubBytes

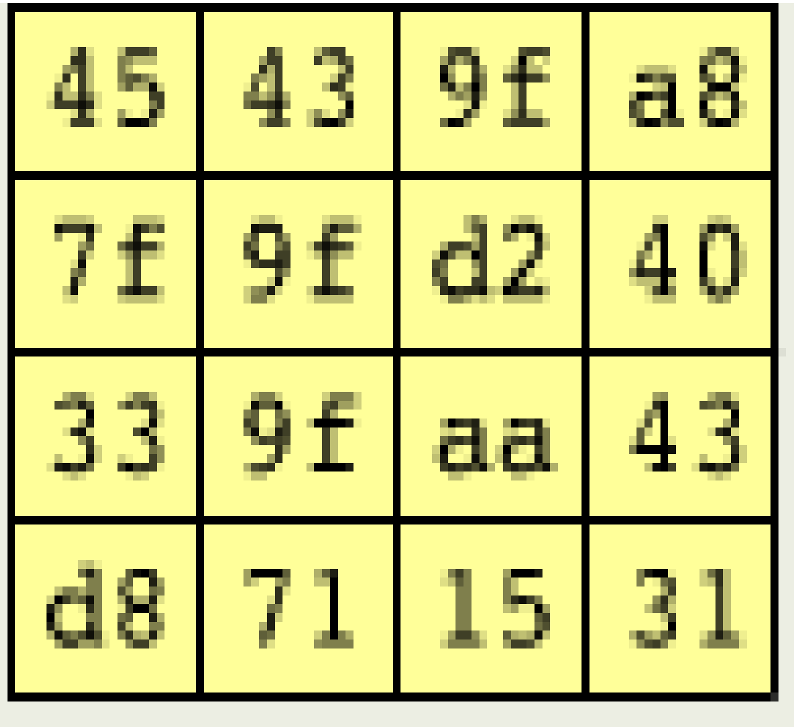


a1	4a	bc	1b
a1	f2	5c	92
b9	66	65	79
4b	e9	e6	43

32	d6	65	af
32	89	4a	4f
56	33	4d	b6
b3	1e	8e	1a

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

ShiftRows

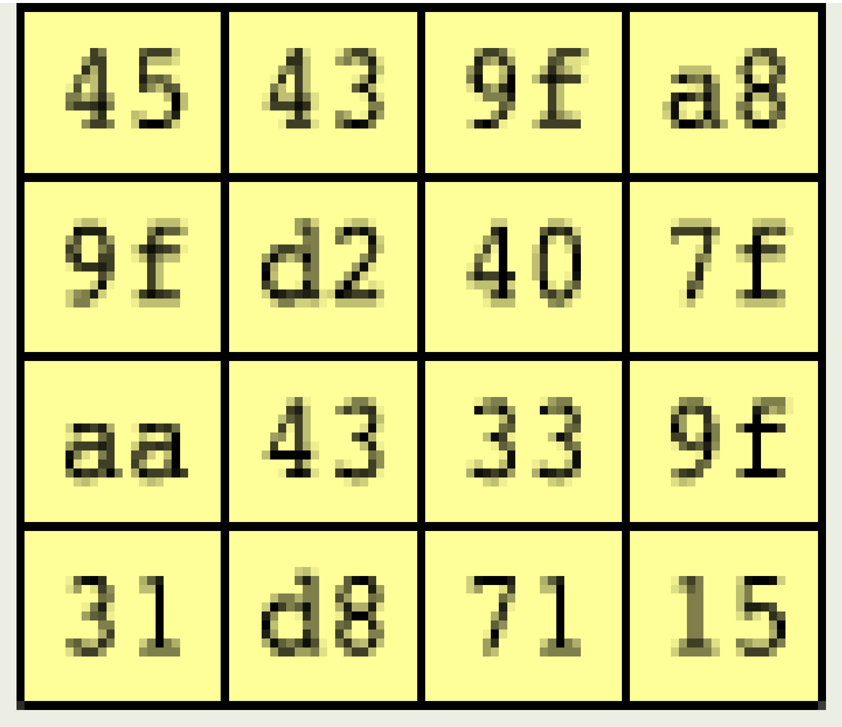


45	43	9f	a8
7f	9f	d2	40
33	9f	aa	43
d8	71	15	31

<-Rotate over 1bytes

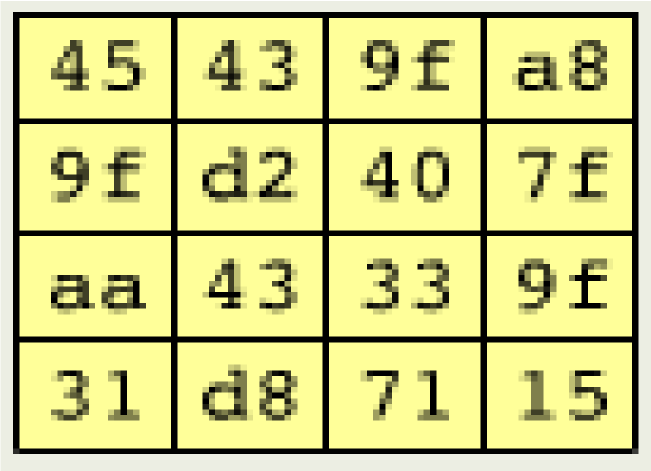
<-Rotate over 2bytes

<-Rotate over 3bytes

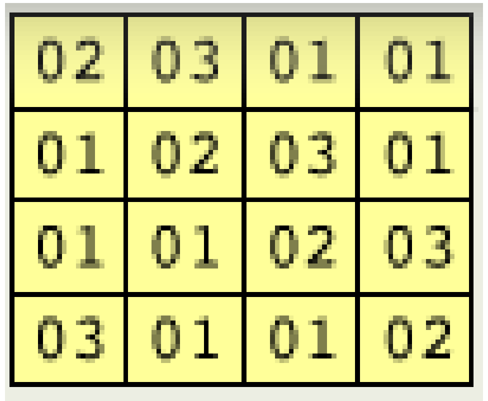


45	43	9f	a8
9f	d2	40	7f
aa	43	33	9f
31	d8	71	15

MixColumns

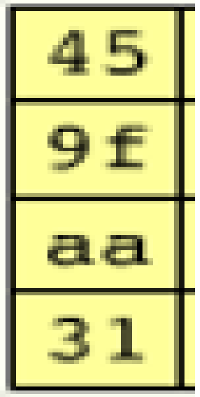


45	43	9f	a8
9f	d2	40	7f
aa	43	33	9f
31	d8	71	15

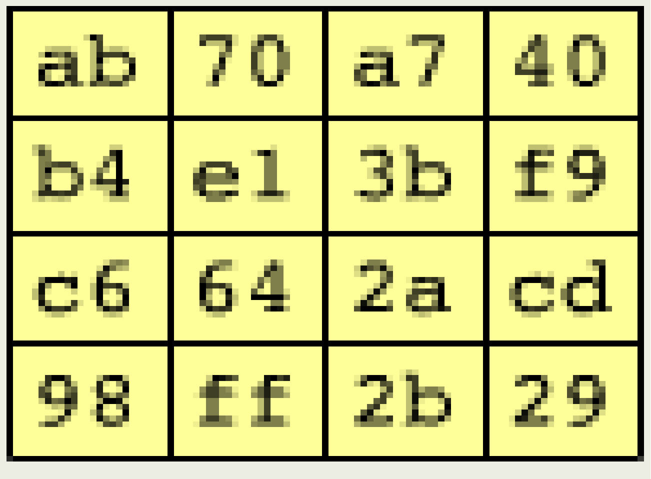


02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

×



45
9f
aa
31



ab	70	a7	40
b4	e1	3b	f9
c6	64	2a	cd
98	ff	2b	29

MixColumns

- **mixColumnsMatrix = 1**

  input 값 그대로 사용

- **mixColumnsMatrix = 2**

  input 값 * 2

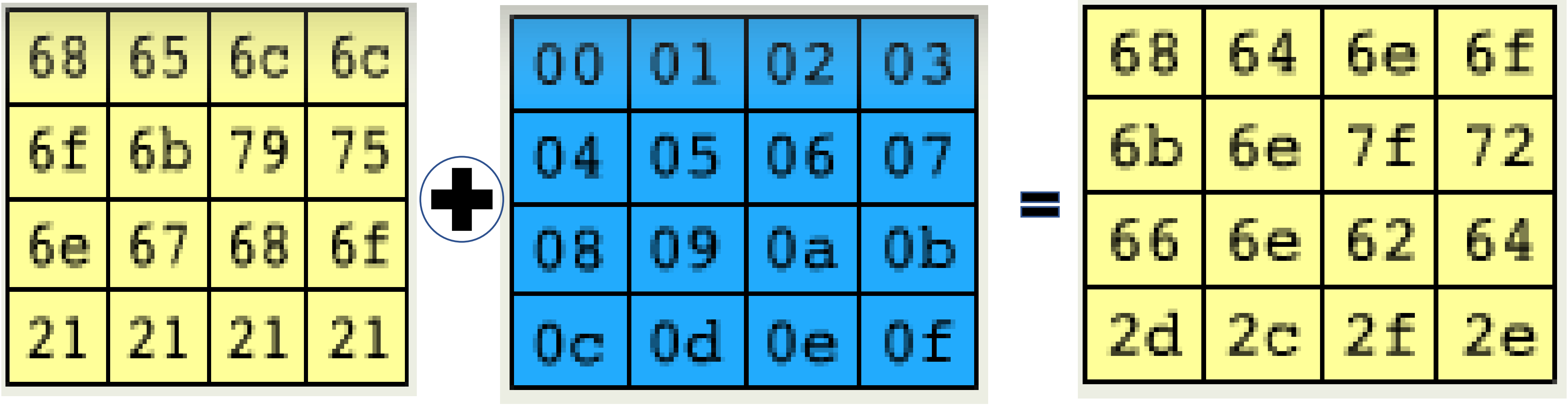
- **mixColumnsMatrix = 3 (0b11 = 0b10 ^ 0b01)**

  (input 값 * 2) ^ (input 값 * 1)

※ input 값에 2를 곱했을때 Overflow가 일어나는 경우 xor 0x1b

->AES에서 사용하는 기약다항식이 x^8+x^4+x^3+x+1이기 때문에
최상위 x^8을 제외하고 바이너리로 표현하면 0001 1011(0x1b)

AddRoundKey



68	65	6c	6c
6f	6b	79	75
6e	67	68	6f
21	21	21	21

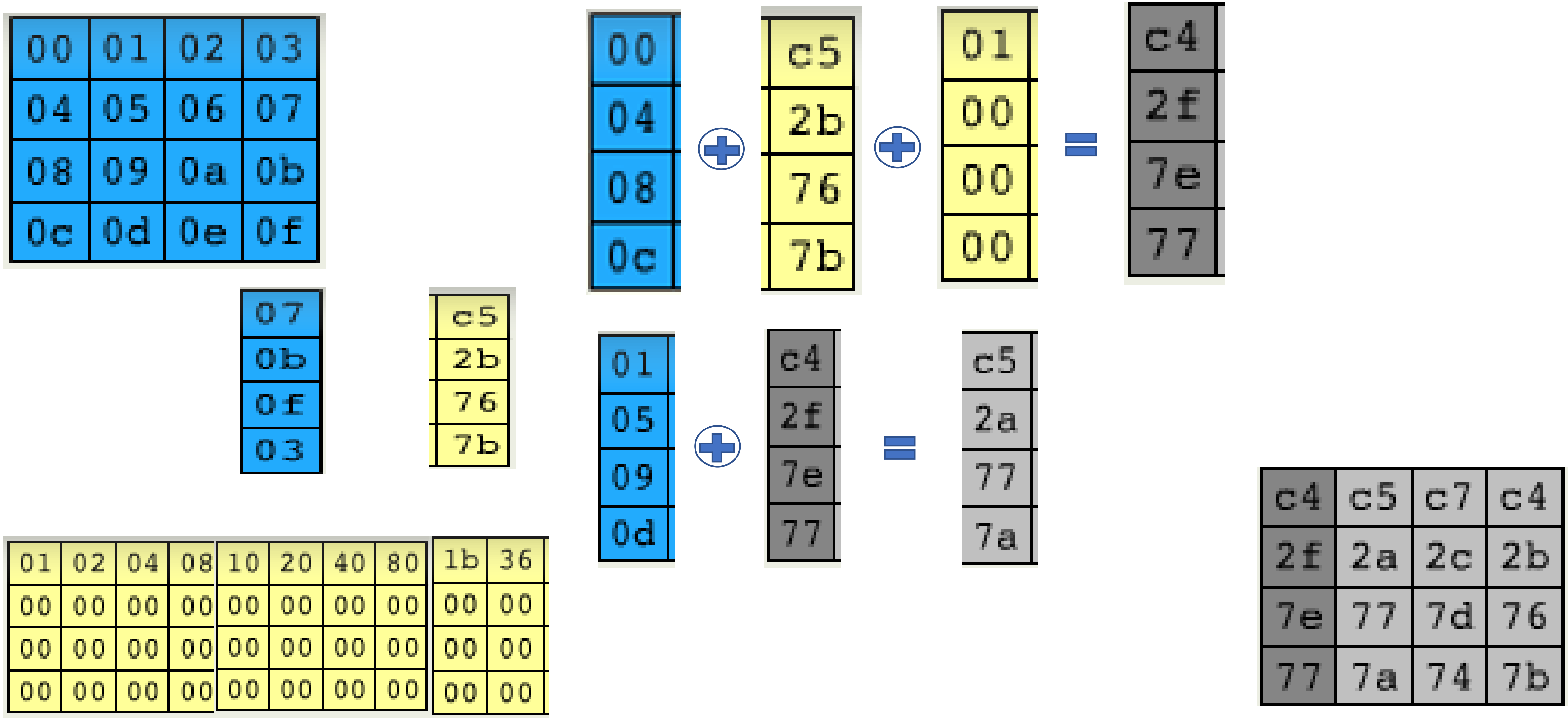
⊕

00	01	02	03
04	05	06	07
08	09	0a	0b
0c	0d	0e	0f

=

68	64	6e	6f
6b	6e	7f	72
66	6e	62	64
2d	2c	2f	2e

KeySchedule



00	01	02	03
04	05	06	07
08	09	0a	0b
0c	0d	0e	0f

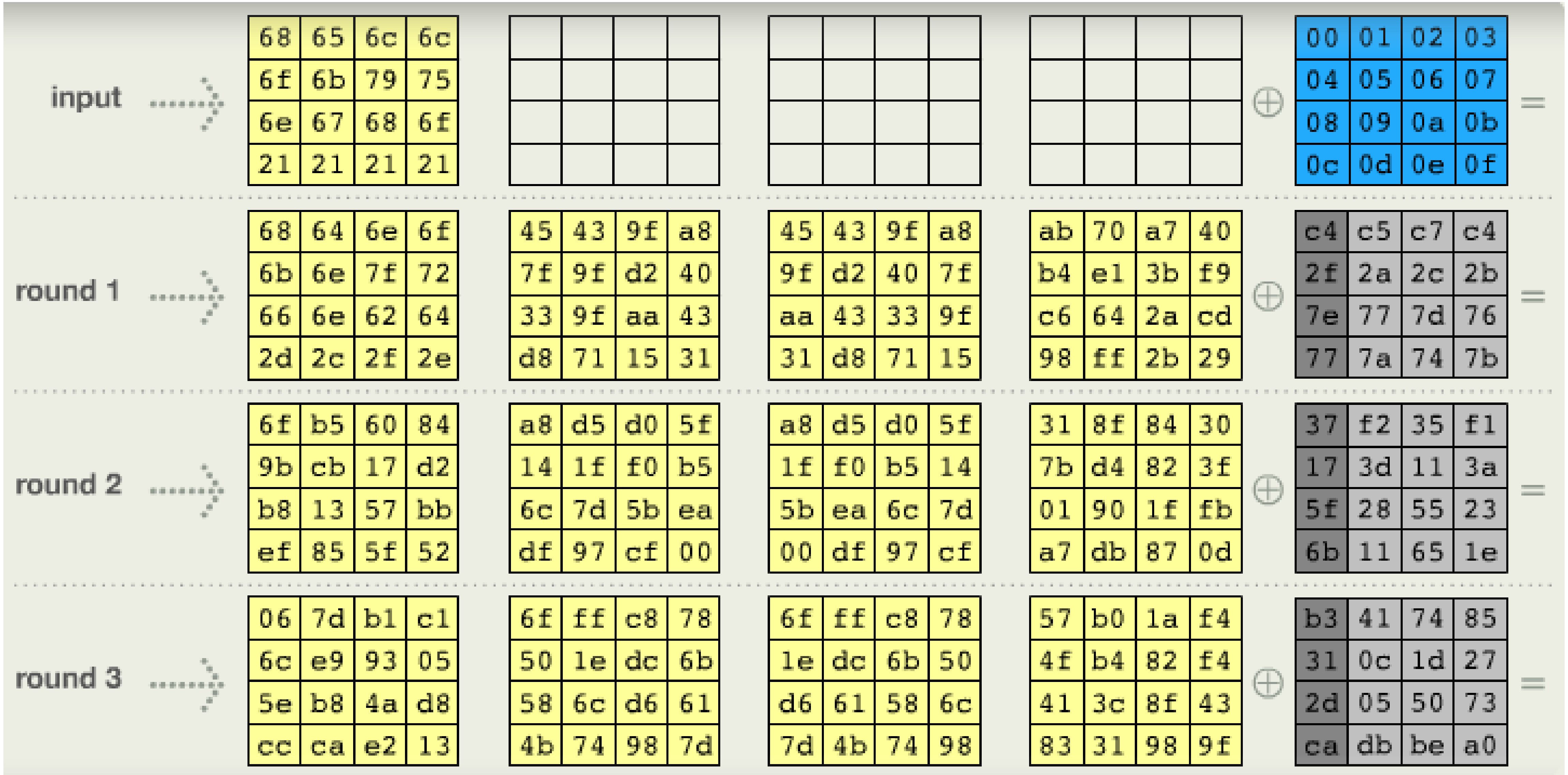
$$\begin{pmatrix}07\\0b\\0f\\03\end{pmatrix} \quad \begin{pmatrix}c5\\2b\\76\\7b\end{pmatrix}$$

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

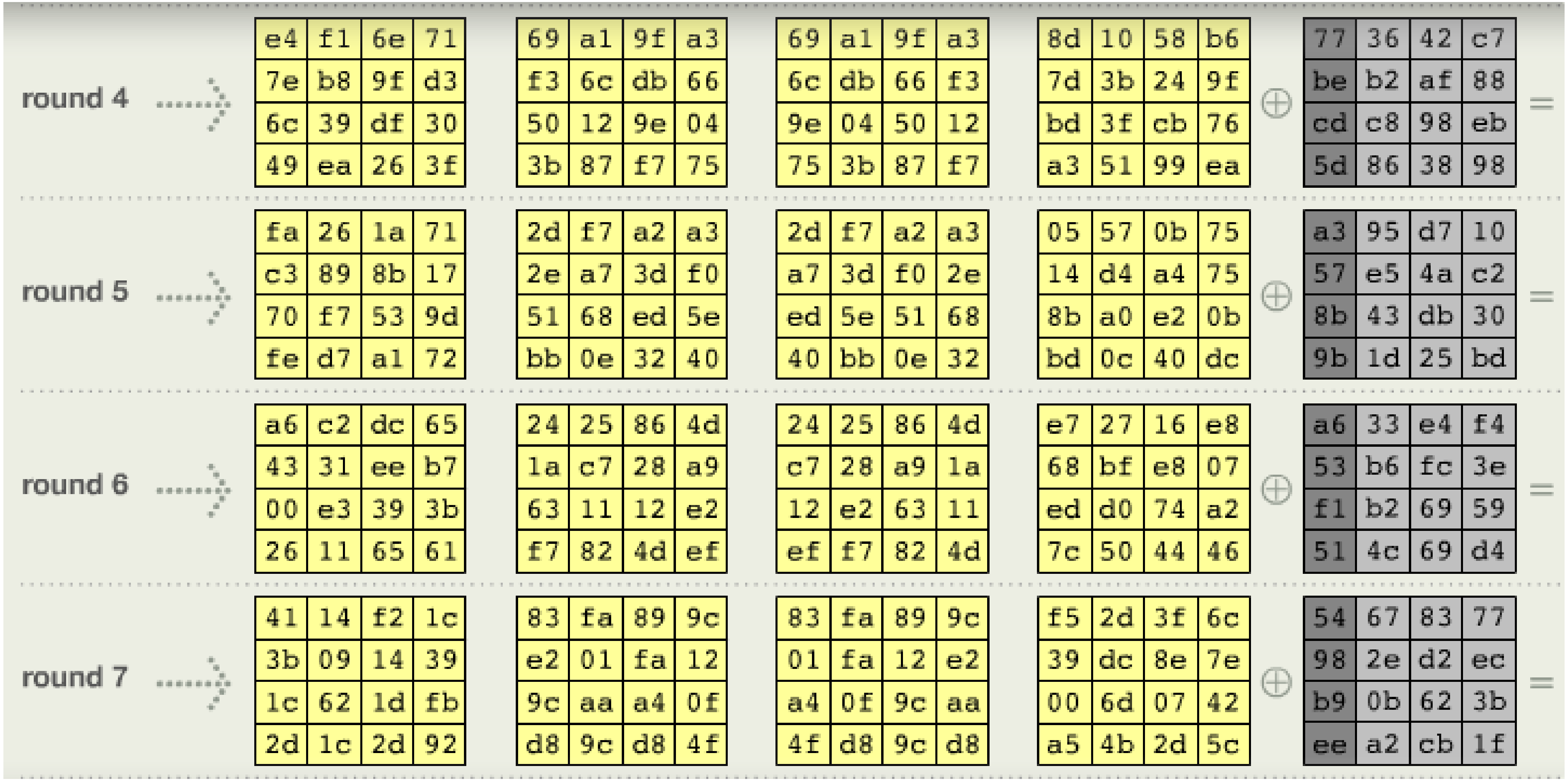
$$\begin{pmatrix}00\\04\\08\\0c\end{pmatrix} \oplus \begin{pmatrix}c5\\2b\\76\\7b\end{pmatrix} \oplus \begin{pmatrix}01\\00\\00\\00\end{pmatrix} = \begin{pmatrix}c4\\2f\\7e\\77\end{pmatrix}$$

$$\begin{pmatrix}01\\05\\09\\0d\end{pmatrix} \oplus \begin{pmatrix}c4\\2f\\7e\\77\end{pmatrix} = \begin{pmatrix}c5\\2a\\77\\7a\end{pmatrix}$$

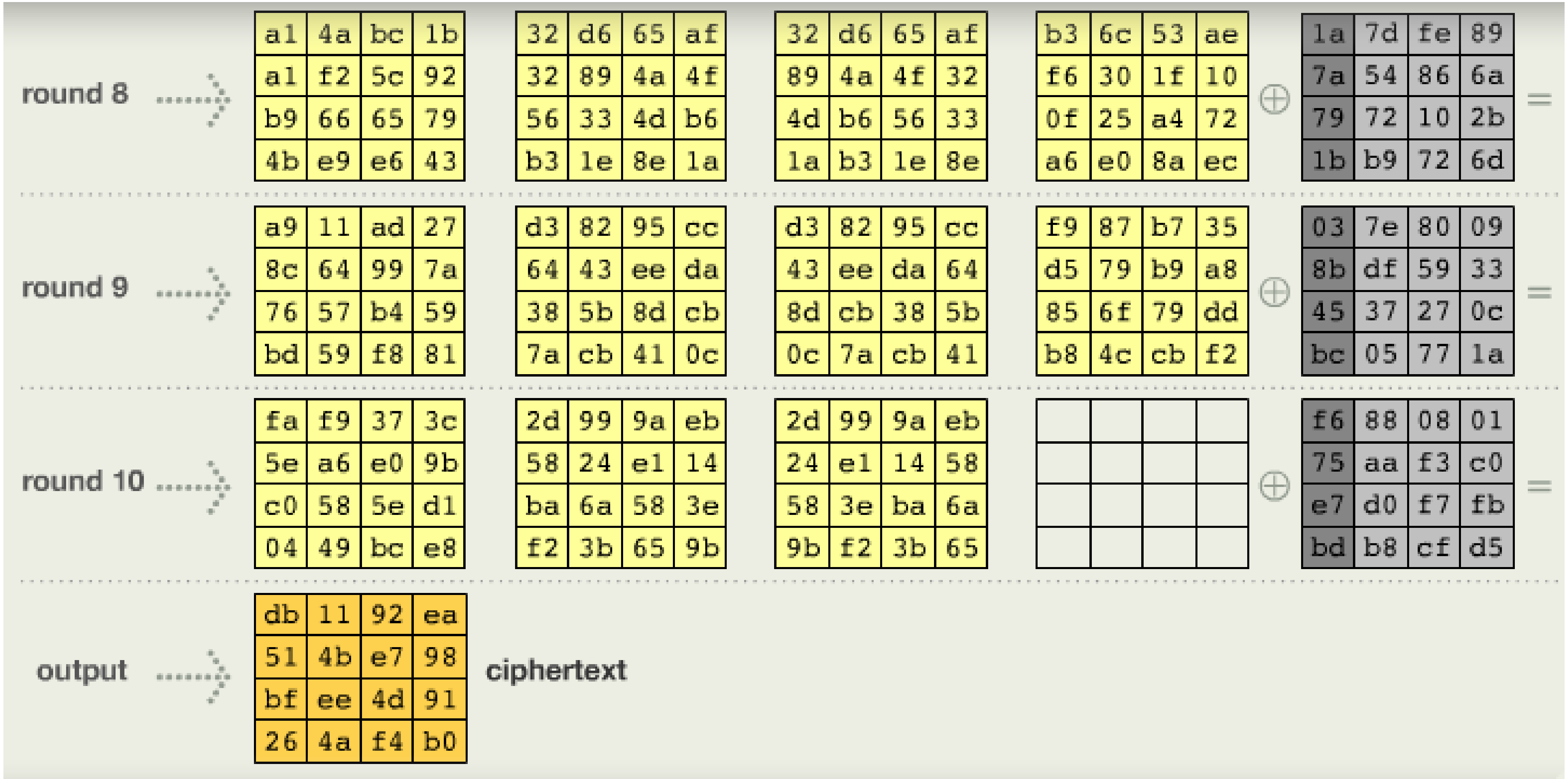
c4	c5	c7	c4
2f	2a	2c	2b
7e	77	7d	76
77	7a	74	7b



input	68	65	6c	6c														00	01	02	03	
	6f	6b	79	75													⊕	04	05	06	07	=
	6e	67	68	6f														08	09	0a	0b	
	21	21	21	21														0c	0d	0e	0f	
round 1	68	64	6e	6f	45	43	9f	a8	45	43	9f	a8	ab	70	a7	40		c4	c5	c7	c4	
	6b	6e	7f	72	7f	9f	d2	40	9f	d2	40	7f	b4	e1	3b	f9	⊕	2f	2a	2c	2b	=
	66	6e	62	64	33	9f	aa	43	aa	43	33	9f	c6	64	2a	cd		7e	77	7d	76	
	2d	2c	2f	2e	d8	71	15	31	31	d8	71	15	98	ff	2b	29		77	7a	74	7b	
round 2	6f	b5	60	84	a8	d5	d0	5f	a8	d5	d0	5f	31	8f	84	30		37	f2	35	f1	
	9b	cb	17	d2	14	1f	f0	b5	1f	f0	b5	14	7b	d4	82	3f	⊕	17	3d	11	3a	=
	b8	13	57	bb	6c	7d	5b	ea	5b	ea	6c	7d	01	90	1f	fb		5f	28	55	23	
	ef	85	5f	52	df	97	cf	00	00	df	97	cf	a7	db	87	0d		6b	11	65	1e	
round 3	06	7d	b1	c1	6f	ff	c8	78	6f	ff	c8	78	57	b0	1a	f4		b3	41	74	85	
	6c	e9	93	05	50	1e	dc	6b	1e	dc	6b	50	4f	b4	82	f4	⊕	31	0c	1d	27	=
	5e	b8	4a	d8	58	6c	d6	61	d6	61	58	6c	41	3c	8f	43		2d	05	50	73	
	cc	ca	e2	13	4b	74	98	7d	7d	4b	74	98	83	31	98	9f		ca	db	be	a0	



round 4	e4	f1	6e	71	69	a1	9f	a3	69	a1	9f	a3	8d	10	58	b6		77	36	42	c7	
	7e	b8	9f	d3	f3	6c	db	66	6c	db	66	f3	7d	3b	24	9f	⊕	be	b2	af	88	=
	6c	39	df	30	50	12	9e	04	9e	04	50	12	bd	3f	cb	76		cd	c8	98	eb	
	49	ea	26	3f	3b	87	f7	75	75	3b	87	f7	a3	51	99	ea		5d	86	38	98	
round 5	fa	26	1a	71	2d	f7	a2	a3	2d	f7	a2	a3	05	57	0b	75		a3	95	d7	10	
	c3	89	8b	17	2e	a7	3d	f0	a7	3d	f0	2e	14	d4	a4	75	⊕	57	e5	4a	c2	=
	70	f7	53	9d	51	68	ed	5e	ed	5e	51	68	8b	a0	e2	0b		8b	43	db	30	
	fe	d7	a1	72	bb	0e	32	40	40	bb	0e	32	bd	0c	40	dc		9b	1d	25	bd	
round 6	a6	c2	dc	65	24	25	86	4d	24	25	86	4d	e7	27	16	e8		a6	33	e4	f4	
	43	31	ee	b7	1a	c7	28	a9	c7	28	a9	1a	68	bf	e8	07	⊕	53	b6	fc	3e	=
	00	e3	39	3b	63	11	12	e2	12	e2	63	11	ed	d0	74	a2		f1	b2	69	59	
	26	11	65	61	f7	82	4d	ef	ef	f7	82	4d	7c	50	44	46		51	4c	69	d4	
round 7	41	14	f2	1c	83	fa	89	9c	83	fa	89	9c	f5	2d	3f	6c		54	67	83	77	
	3b	09	14	39	e2	01	fa	12	01	fa	12	e2	39	dc	8e	7e	⊕	98	2e	d2	ec	=
	1c	62	1d	fb	9c	aa	a4	0f	a4	0f	9c	aa	00	6d	07	42		b9	0b	62	3b	
	2d	1c	2d	92	d8	9c	d8	4f	4f	d8	9c	d8	a5	4b	2d	5c		ee	a2	cb	1f	



round 8

a1	4a	bc	1b		32	d6	65	af		32	d6	65	af		b3	6c	53	ae		1a	7d	fe	89	
a1	f2	5c	92		32	89	4a	4f		89	4a	4f	32		f6	30	1f	10	⊕	7a	54	86	6a	=
b9	66	65	79		56	33	4d	b6		4d	b6	56	33		0f	25	a4	72		79	72	10	2b	
4b	e9	e6	43		b3	1e	8e	1a		1a	b3	1e	8e		a6	e0	8a	ec		1b	b9	72	6d	

round 9

a9	11	ad	27		d3	82	95	cc		d3	82	95	cc		f9	87	b7	35		03	7e	80	09	
8c	64	99	7a		64	43	ee	da		43	ee	da	64		d5	79	b9	a8	⊕	8b	df	59	33	=
76	57	b4	59		38	5b	8d	cb		8d	cb	38	5b		85	6f	79	dd		45	37	27	0c	
bd	59	f8	81		7a	cb	41	0c		0c	7a	cb	41		b8	4c	cb	f2		bc	05	77	1a	

round 10

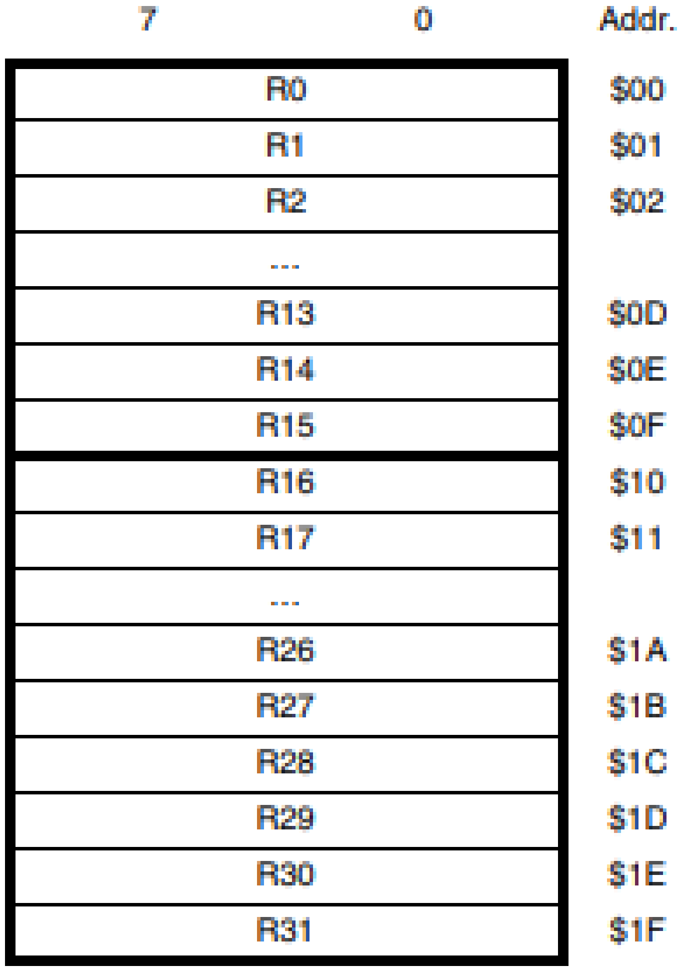
fa	f9	37	3c		2d	99	9a	eb		2d	99	9a	eb							f6	88	08	01	
5e	a6	e0	9b		58	24	e1	14		24	e1	14	58						⊕	75	aa	f3	c0	=
c0	58	5e	d1		ba	6a	58	3e		58	3e	ba	6a							e7	d0	f7	fb	
04	49	bc	e8		f2	3b	65	9b		9b	f2	3b	65							bd	b8	cf	d5	

output

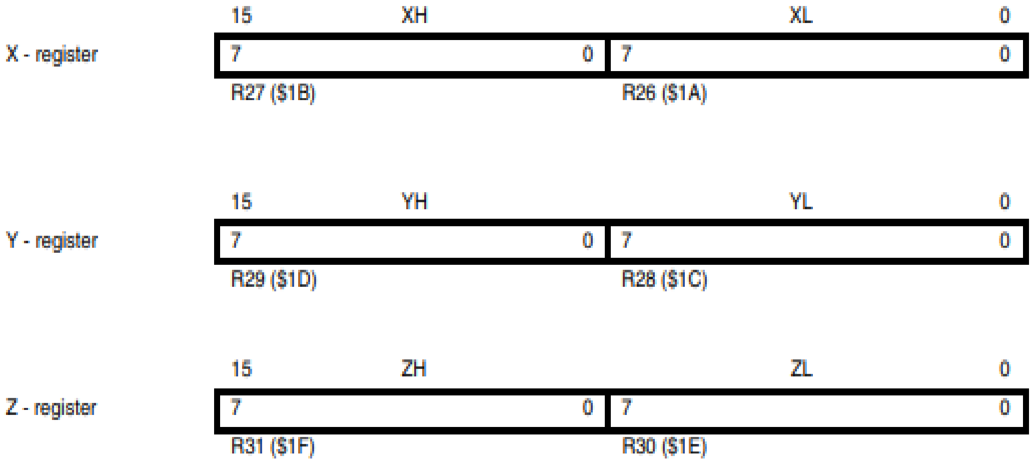
db	11	92	ea
51	4b	e7	98
bf	ee	4d	91
26	4a	f4	b0

ciphertext

2. 기본적인 AVR assembly



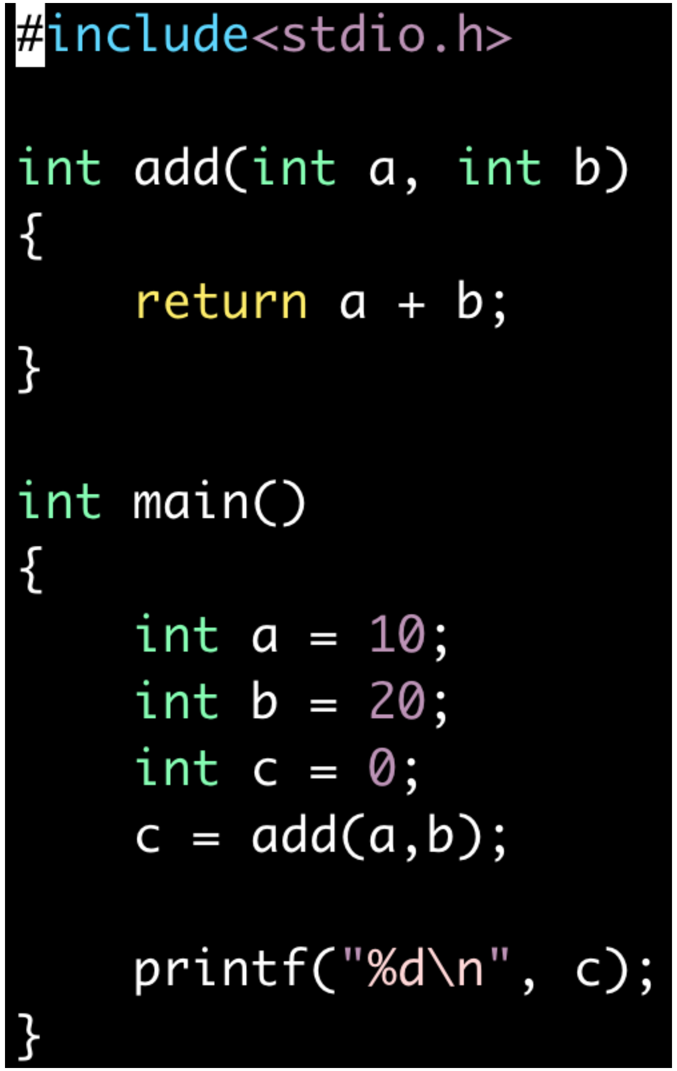
7 0	Addr.
R0	$00
R1	$01
R2	$02
...	
R13	$0D
R14	$0E
R15	$0F
R16	$10
R17	$11
...	
R26	$1A
R27	$1B
R28	$1C
R29	$1D
R30	$1E
R31	$1F



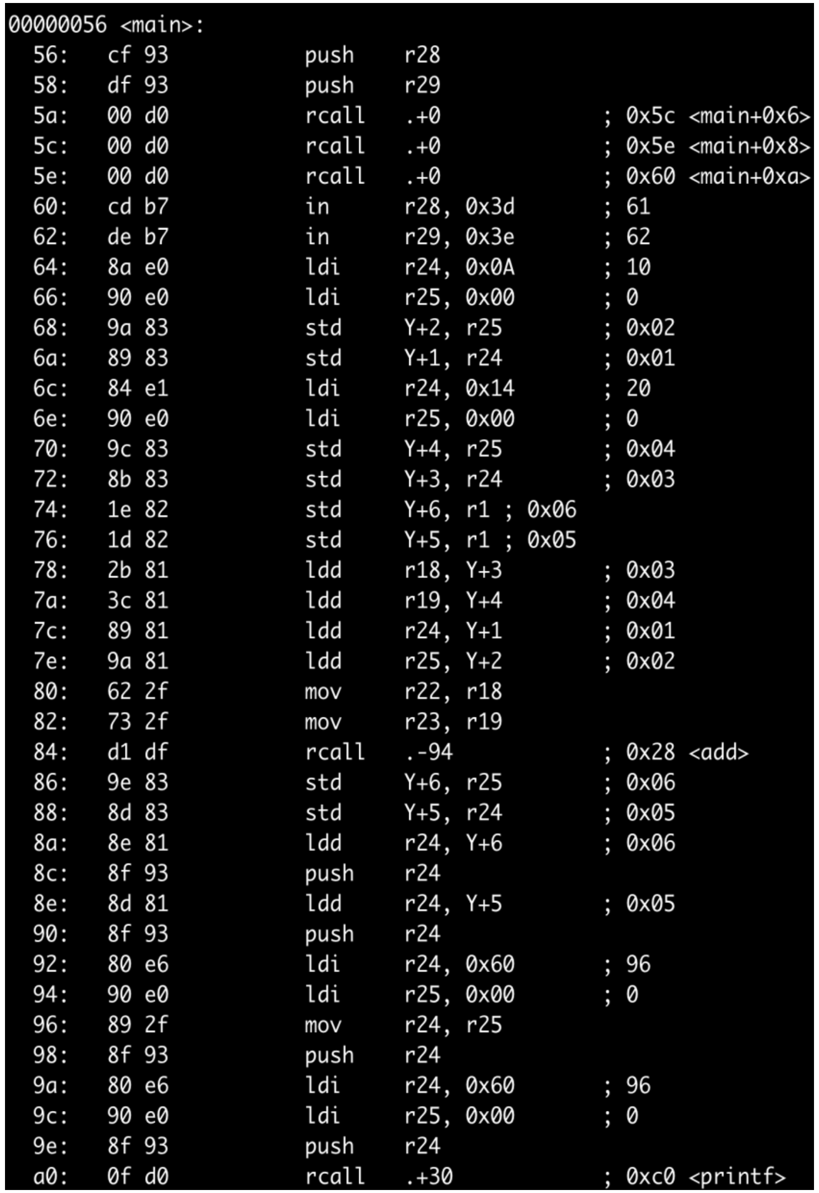
함수의 매개 변수 -> R24, R25
R22, R23 ...

함수의 리턴 값 -> R24, R25

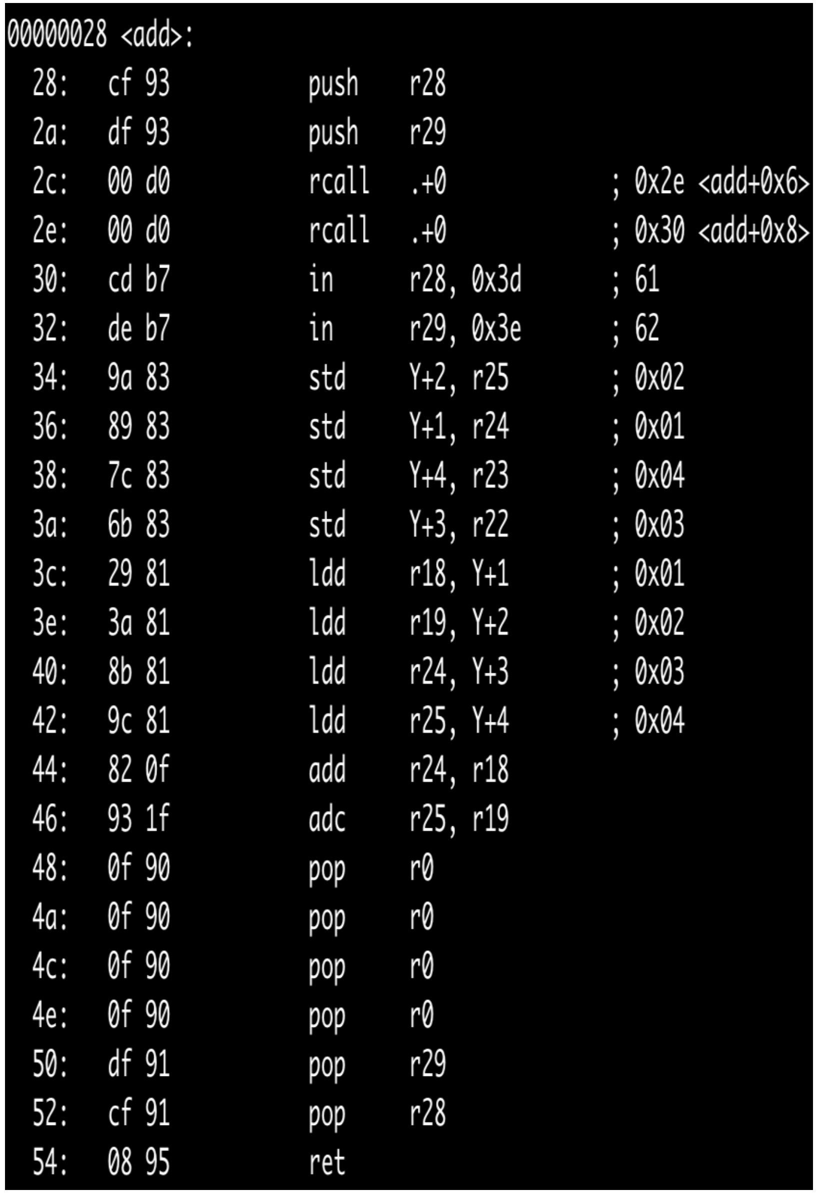
R1 -> 항상 0 (Zero)



```c
#include<stdio.h>

int add(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 20;
    int c = 0;
    c = add(a,b);

    printf("%d\n", c);
}
```

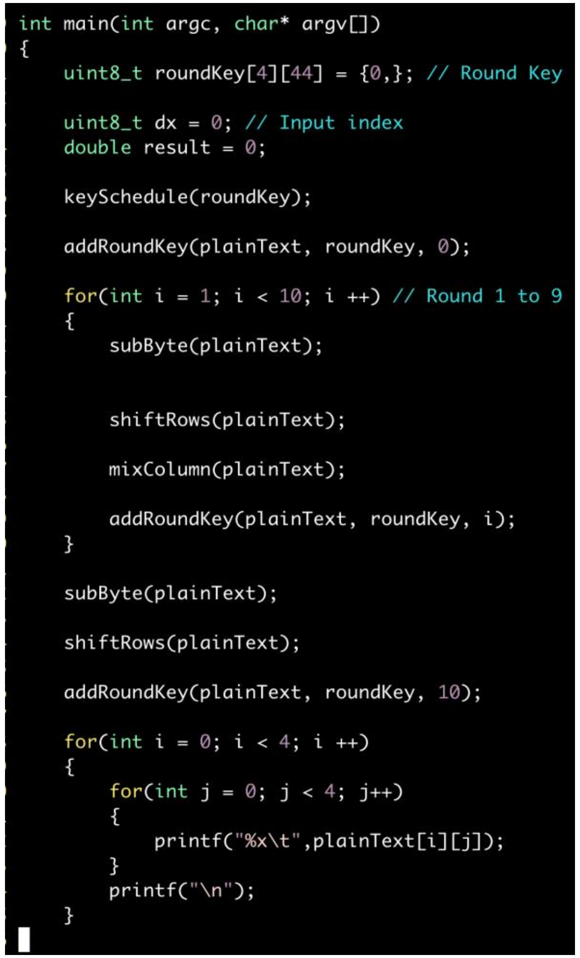


```
00000056 <main>:
  56:   cf 93           push    r28
  58:   df 93           push    r29
  5a:   00 d0           rcall   .+0             ; 0x5c <main+0x6>
  5c:   00 d0           rcall   .+0             ; 0x5e <main+0x8>
  5e:   00 d0           rcall   .+0             ; 0x60 <main+0xa>
  60:   cd b7           in      r28, 0x3d       ; 61
  62:   de b7           in      r29, 0x3e       ; 62
  64:   8a e0           ldi     r24, 0x0A       ; 10
  66:   90 e0           ldi     r25, 0x00       ; 0
  68:   9a 83           std     Y+2, r25        ; 0x02
  6a:   89 83           std     Y+1, r24        ; 0x01
  6c:   84 e1           ldi     r24, 0x14       ; 20
  6e:   90 e0           ldi     r25, 0x00       ; 0
  70:   9c 83           std     Y+4, r25        ; 0x04
  72:   8b 83           std     Y+3, r24        ; 0x03
  74:   1e 82           std     Y+6, r1 ; 0x06
  76:   1d 82           std     Y+5, r1 ; 0x05
  78:   2b 81           ldd     r18, Y+3        ; 0x03
  7a:   3c 81           ldd     r19, Y+4        ; 0x04
  7c:   89 81           ldd     r24, Y+1        ; 0x01
  7e:   9a 81           ldd     r25, Y+2        ; 0x02
  80:   62 2f           mov     r22, r18
  82:   73 2f           mov     r23, r19
  84:   d1 df           rcall   .-94            ; 0x28 <add>
  86:   9e 83           std     Y+6, r25        ; 0x06
  88:   8d 83           std     Y+5, r24        ; 0x05
  8a:   8e 81           ldd     r24, Y+6        ; 0x06
  8c:   8f 93           push    r24
  8e:   8d 81           ldd     r24, Y+5        ; 0x05
  90:   8f 93           push    r24
  92:   80 e6           ldi     r24, 0x60       ; 96
  94:   90 e0           ldi     r25, 0x00       ; 0
  96:   89 2f           mov     r24, r25
  98:   8f 93           push    r24
  9a:   80 e6           ldi     r24, 0x60       ; 96
  9c:   90 e0           ldi     r25, 0x00       ; 0
  9e:   8f 93           push    r24
  a0:   0f d0           rcall   .+30            ; 0xc0 <printf>
```

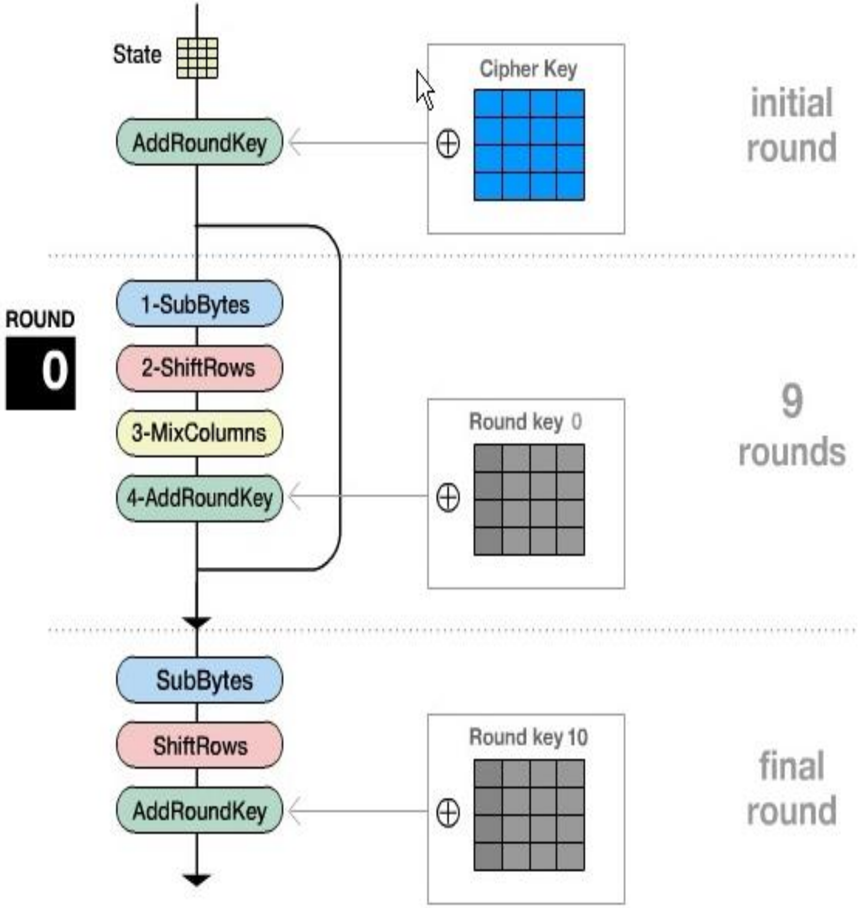


```
00000028 <add>:
  28:   cf 93           push    r28
  2a:   df 93           push    r29
  2c:   00 d0           rcall   .+0             ; 0x2e <add+0x6>
  2e:   00 d0           rcall   .+0             ; 0x30 <add+0x8>
  30:   cd b7           in      r28, 0x3d       ; 61
  32:   de b7           in      r29, 0x3e       ; 62
  34:   9a 83           std     Y+2, r25        ; 0x02
  36:   89 83           std     Y+1, r24        ; 0x01
  38:   7c 83           std     Y+4, r23        ; 0x04
  3a:   6b 83           std     Y+3, r22        ; 0x03
  3c:   29 81           ldd     r18, Y+1        ; 0x01
  3e:   3a 81           ldd     r19, Y+2        ; 0x02
  40:   8b 81           ldd     r24, Y+3        ; 0x03
  42:   9c 81           ldd     r25, Y+4        ; 0x04
  44:   82 0f           add     r24, r18
  46:   93 1f           adc     r25, r19
  48:   0f 90           pop     r0
  4a:   0f 90           pop     r0
  4c:   0f 90           pop     r0
  4e:   0f 90           pop     r0
  50:   df 91           pop     r29
  52:   cf 91           pop     r28
  54:   08 95           ret
```

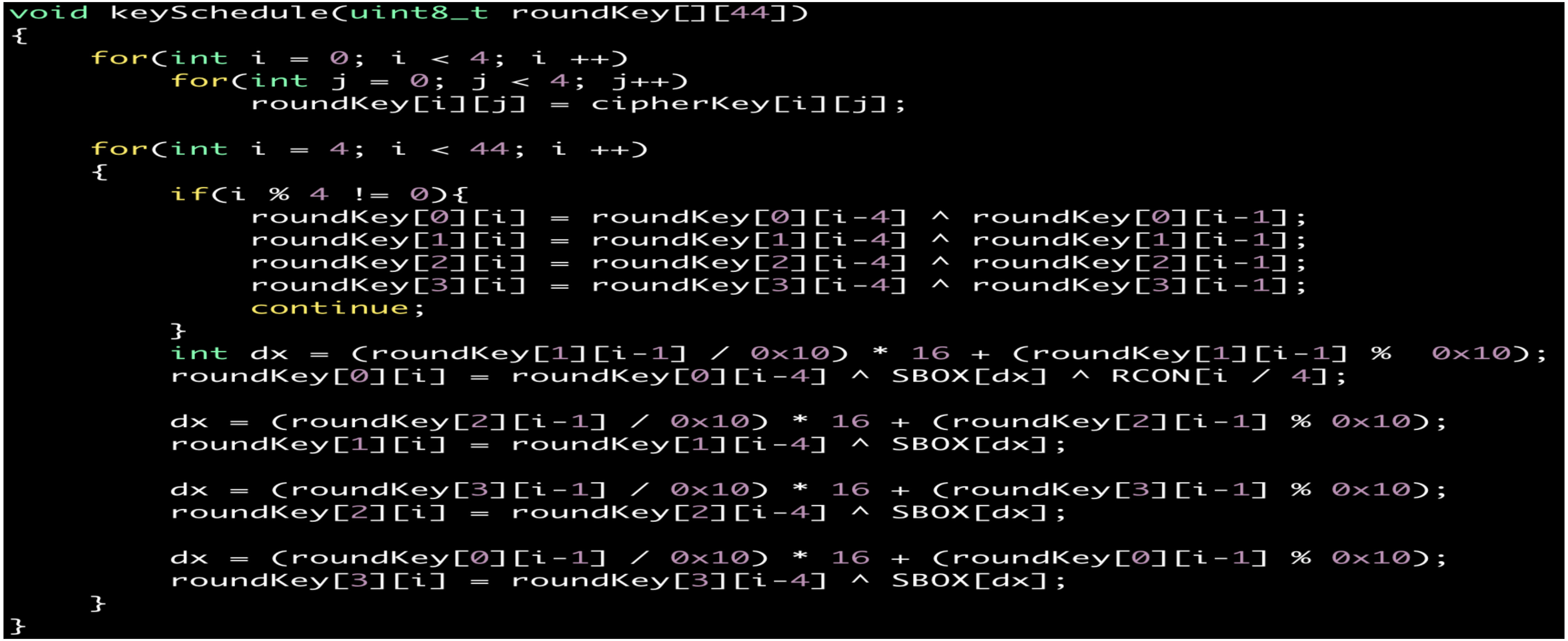
3. C & inline assembly를 이용한 AES 구현



```
int main(int argc, char* argv[])
{
    uint8_t roundKey[4][44] = {0,}; // Round Key

    uint8_t dx = 0; // Input index
    double result = 0;

    keySchedule(roundKey);

    addRoundKey(plainText, roundKey, 0);

    for(int i = 1; i < 10; i ++) // Round 1 to 9
    {
        subByte(plainText);


        shiftRows(plainText);

        mixColumn(plainText);

        addRoundKey(plainText, roundKey, i);
    }

    subByte(plainText);

    shiftRows(plainText);

    addRoundKey(plainText, roundKey, 10);

    for(int i = 0; i < 4; i ++)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%x\t",plainText[i][j]);
        }
        printf("\n");
    }
```

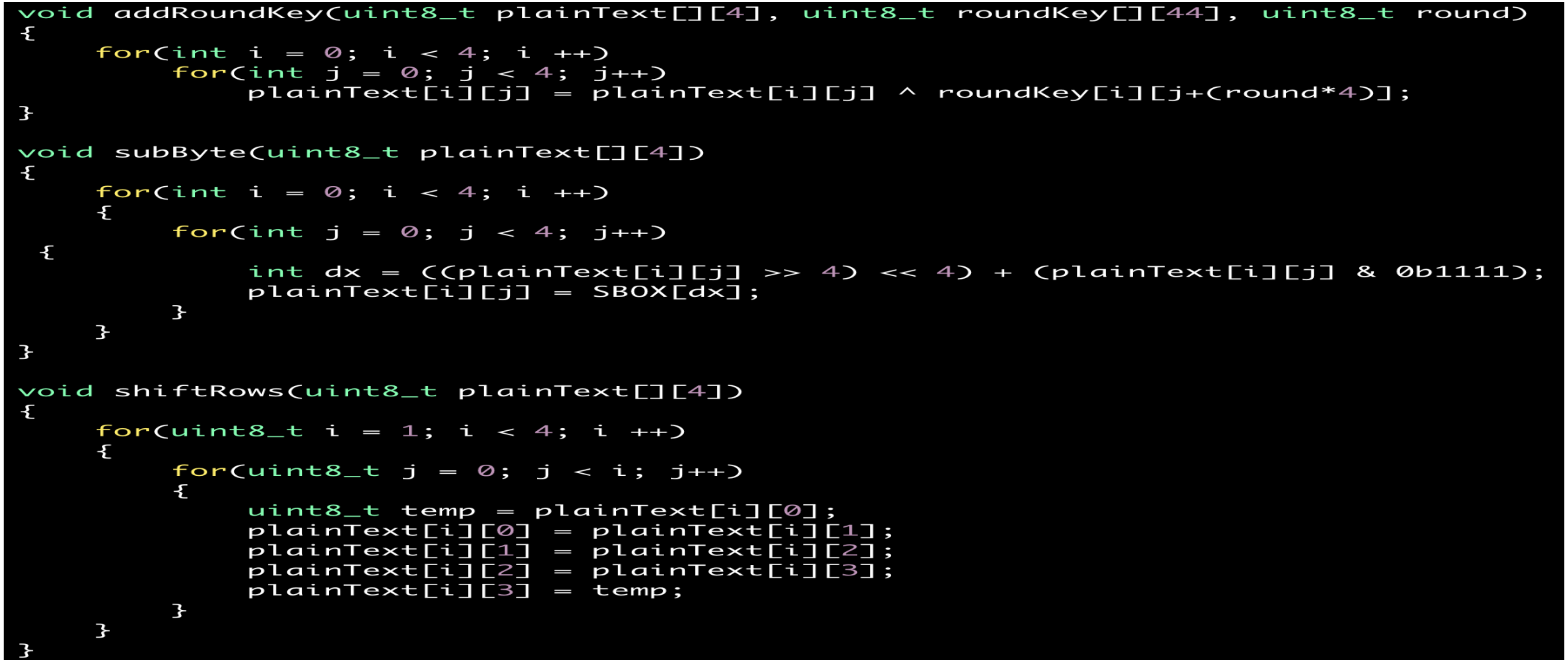


KeySchedule



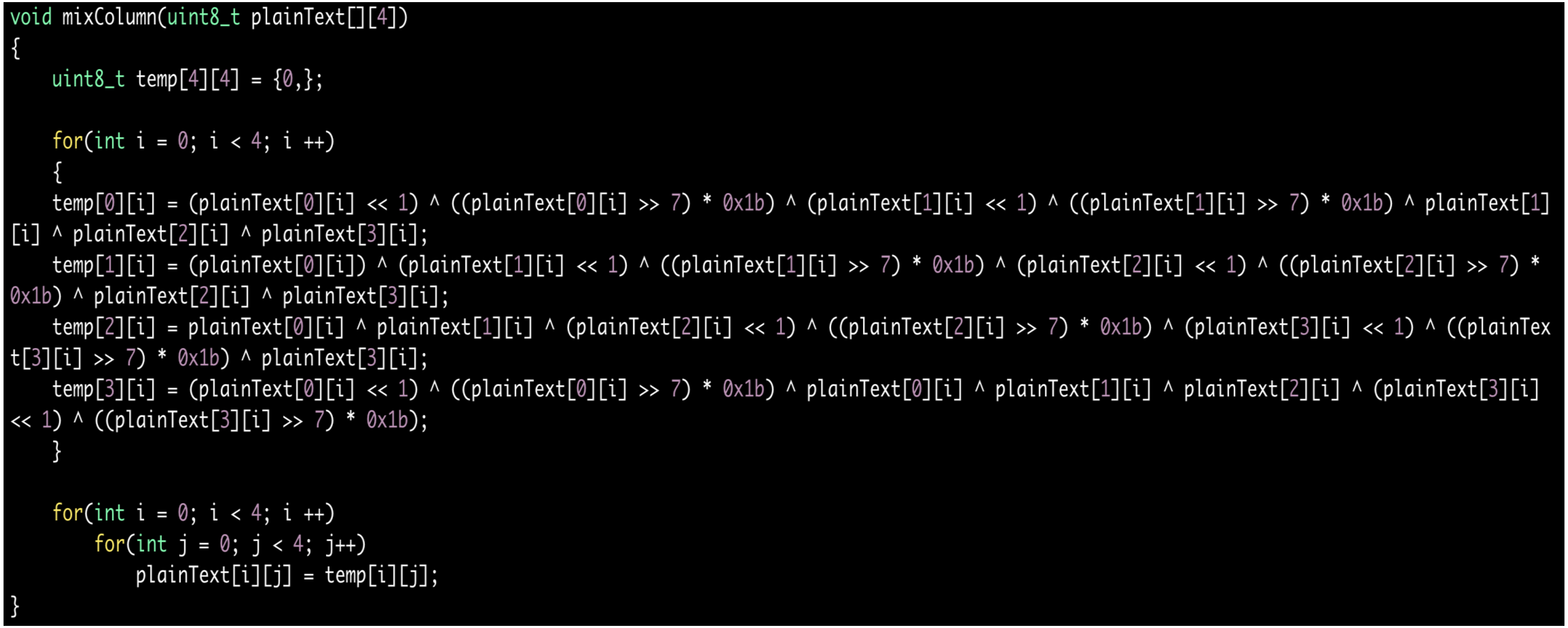
```
void keySchedule(uint8_t roundKey[][44])
{
    for(int i = 0; i < 4; i ++)
        for(int j = 0; j < 4; j++)
            roundKey[i][j] = cipherKey[i][j];

    for(int i = 4; i < 44; i ++)
    {
        if(i % 4 != 0){
            roundKey[0][i] = roundKey[0][i-4] ^ roundKey[0][i-1];
            roundKey[1][i] = roundKey[1][i-4] ^ roundKey[1][i-1];
            roundKey[2][i] = roundKey[2][i-4] ^ roundKey[2][i-1];
            roundKey[3][i] = roundKey[3][i-4] ^ roundKey[3][i-1];
            continue;
        }
        int dx = (roundKey[1][i-1] / 0x10) * 16 + (roundKey[1][i-1] %  0x10);
        roundKey[0][i] = roundKey[0][i-4] ^ SBOX[dx] ^ RCON[i / 4];

        dx = (roundKey[2][i-1] / 0x10) * 16 + (roundKey[2][i-1] % 0x10);
        roundKey[1][i] = roundKey[1][i-4] ^ SBOX[dx];

        dx = (roundKey[3][i-1] / 0x10) * 16 + (roundKey[3][i-1] % 0x10);
        roundKey[2][i] = roundKey[2][i-4] ^ SBOX[dx];

        dx = (roundKey[0][i-1] / 0x10) * 16 + (roundKey[0][i-1] % 0x10);
        roundKey[3][i] = roundKey[3][i-4] ^ SBOX[dx];
    }
}
```

AddRoundKey & SubBytes & ShiftRows



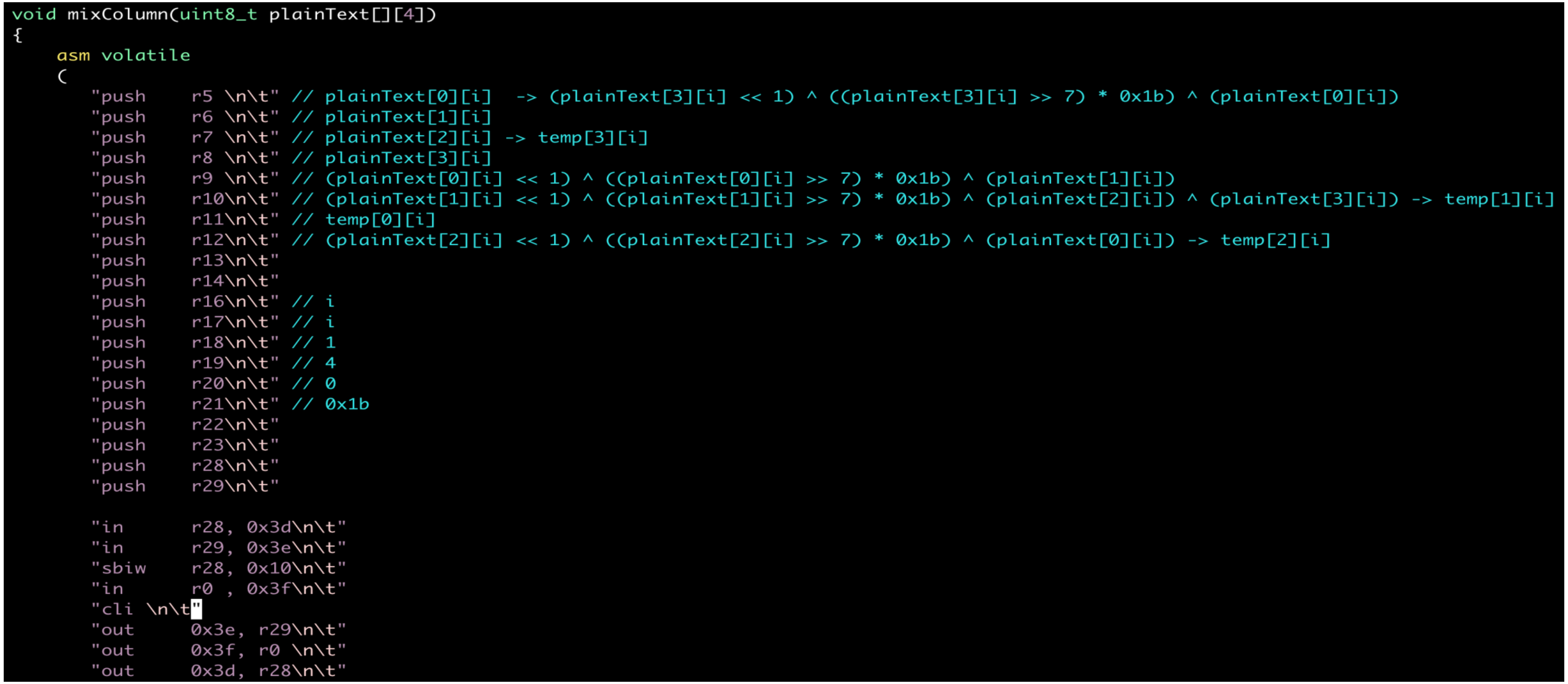
```
void addRoundKey(uint8_t plainText[][4], uint8_t roundKey[][44], uint8_t round)
{
        for(int i = 0; i < 4; i ++)
                for(int j = 0; j < 4; j++)
                        plainText[i][j] = plainText[i][j] ^ roundKey[i][j+(round*4)];
}

void subByte(uint8_t plainText[][4])
{
        for(int i = 0; i < 4; i ++)
        {
                for(int j = 0; j < 4; j++)
  {
                        int dx = ((plainText[i][j] >> 4) << 4) + (plainText[i][j] & 0b1111);
                        plainText[i][j] = SBOX[dx];
                }
        }
}

void shiftRows(uint8_t plainText[][4])
{
        for(uint8_t i = 1; i < 4; i ++)
        {
                for(uint8_t j = 0; j < i; j++)
                {
                        uint8_t temp = plainText[i][0];
                        plainText[i][0] = plainText[i][1];
                        plainText[i][1] = plainText[i][2];
                        plainText[i][2] = plainText[i][3];
                        plainText[i][3] = temp;
                }
        }
}
```

MixColumns (C Ver)

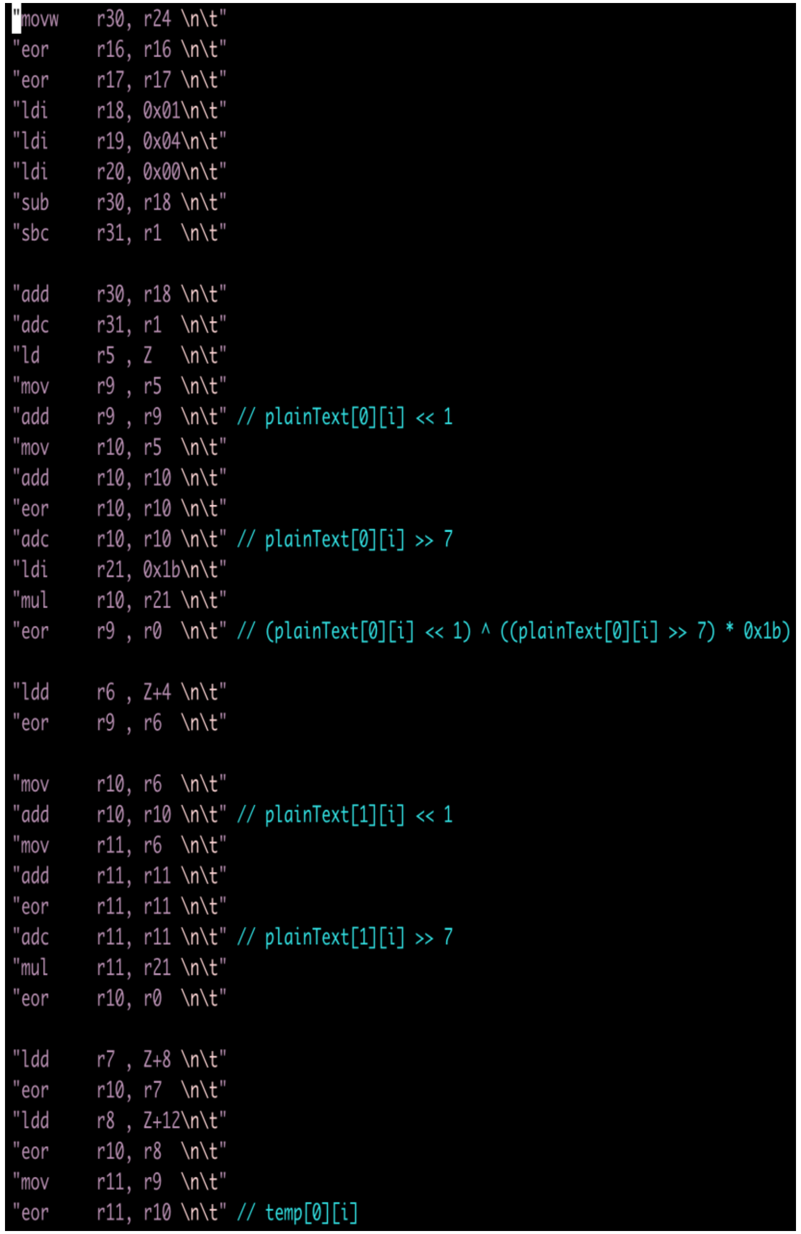


```c
void mixColumn(uint8_t plainText[][4])
{
    uint8_t temp[4][4] = {0,};

    for(int i = 0; i < 4; i ++)
    {
    temp[0][i] = (plainText[0][i] << 1) ^ ((plainText[0][i] >> 7) * 0x1b) ^ (plainText[1][i] << 1) ^ ((plainText[1][i] >> 7) * 0x1b) ^ plainText[1][i] ^ plainText[2][i] ^ plainText[3][i];
    temp[1][i] = (plainText[0][i]) ^ (plainText[1][i] << 1) ^ ((plainText[1][i] >> 7) * 0x1b) ^ (plainText[2][i] << 1) ^ ((plainText[2][i] >> 7) * 0x1b) ^ plainText[2][i] ^ plainText[3][i];
    temp[2][i] = plainText[0][i] ^ plainText[1][i] ^ (plainText[2][i] << 1) ^ ((plainText[2][i] >> 7) * 0x1b) ^ (plainText[3][i] << 1) ^ ((plainText[3][i] >> 7) * 0x1b) ^ plainText[3][i];
    temp[3][i] = (plainText[0][i] << 1) ^ ((plainText[0][i] >> 7) * 0x1b) ^ plainText[0][i] ^ plainText[1][i] ^ plainText[2][i] ^ (plainText[3][i] << 1) ^ ((plainText[3][i] >> 7) * 0x1b);
    }

    for(int i = 0; i < 4; i ++)
        for(int j = 0; j < 4; j++)
            plainText[i][j] = temp[i][j];
}
```

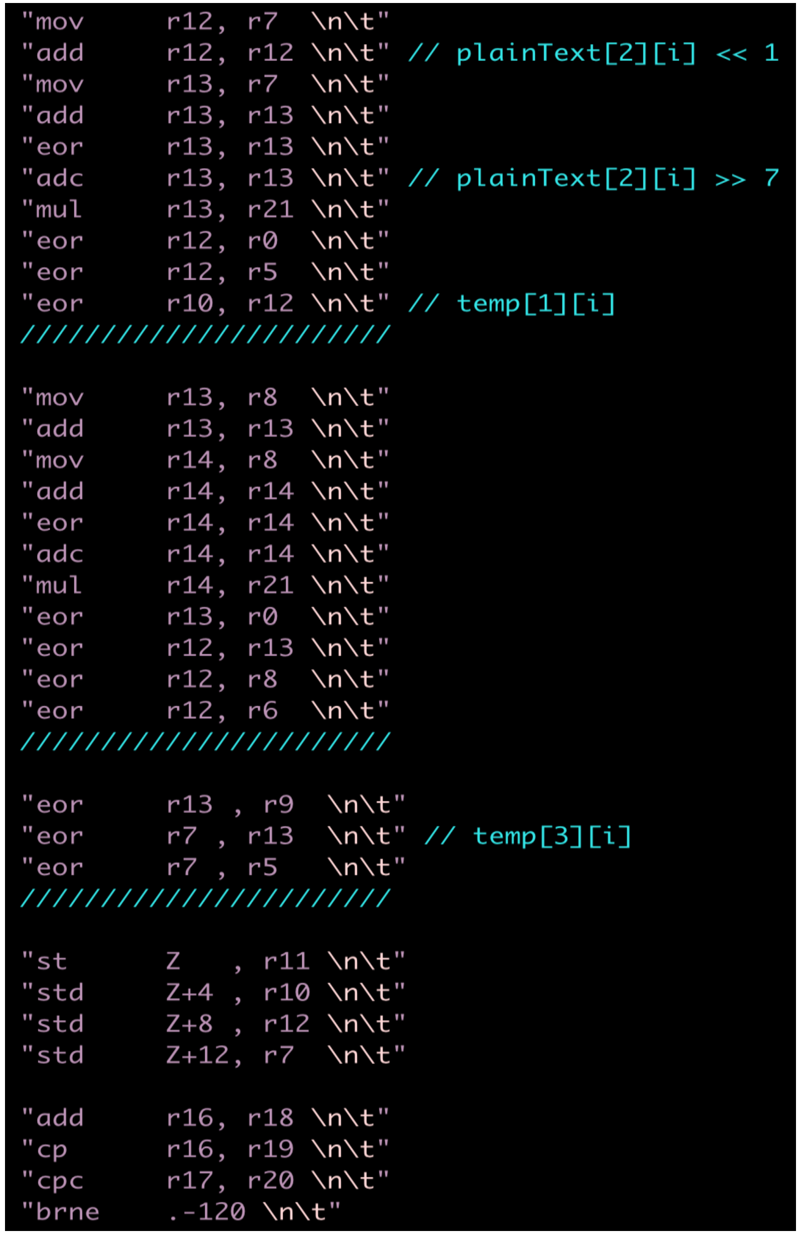
MixColumns (Asm Ver)



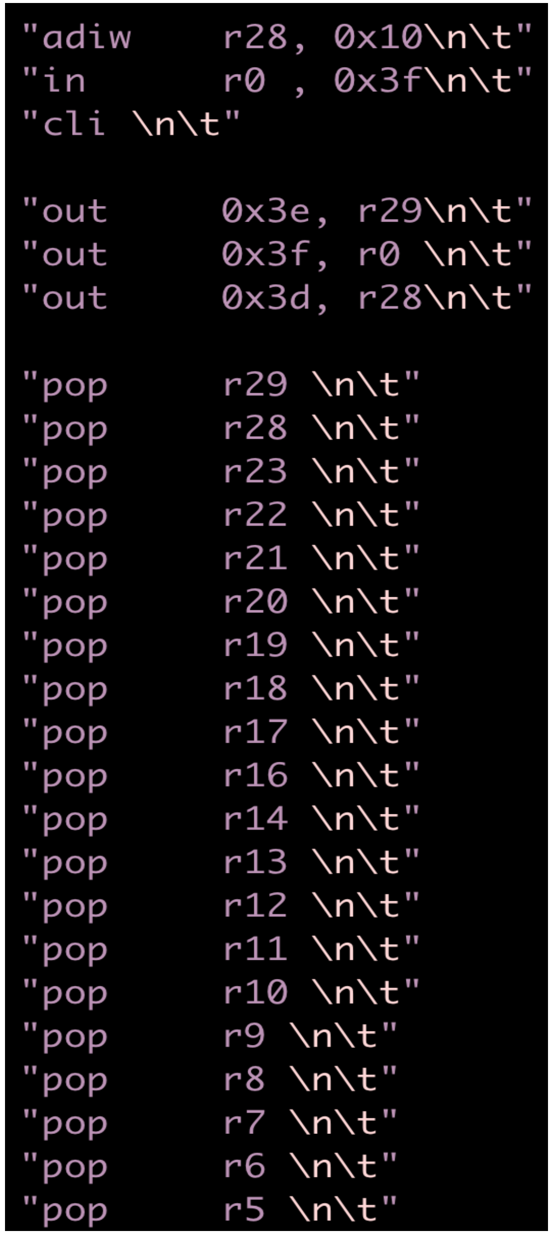
```
void mixColumn(uint8_t plainText[][4])
{
    asm volatile
    (
       "push    r5 \n\t" // plainText[0][i]  -> (plainText[3][i] << 1) ^ ((plainText[3][i] >> 7) * 0x1b) ^ (plainText[0][i])
       "push    r6 \n\t" // plainText[1][i]
       "push    r7 \n\t" // plainText[2][i] -> temp[3][i]
       "push    r8 \n\t" // plainText[3][i]
       "push    r9 \n\t" // (plainText[0][i] << 1) ^ ((plainText[0][i] >> 7) * 0x1b) ^ (plainText[1][i])
       "push    r10\n\t" // (plainText[1][i] << 1) ^ ((plainText[1][i] >> 7) * 0x1b) ^ (plainText[2][i]) ^ (plainText[3][i]) -> temp[1][i]
       "push    r11\n\t" // temp[0][i]
       "push    r12\n\t" // (plainText[2][i] << 1) ^ ((plainText[2][i] >> 7) * 0x1b) ^ (plainText[0][i]) -> temp[2][i]
       "push    r13\n\t"
       "push    r14\n\t"
       "push    r16\n\t" // i
       "push    r17\n\t" // i
       "push    r18\n\t" // 1
       "push    r19\n\t" // 4
       "push    r20\n\t" // 0
       "push    r21\n\t" // 0x1b
       "push    r22\n\t"
       "push    r23\n\t"
       "push    r28\n\t"
       "push    r29\n\t"

       "in      r28, 0x3d\n\t"
       "in      r29, 0x3e\n\t"
       "sbiw    r28, 0x10\n\t"
       "in      r0 , 0x3f\n\t"
       "cli \n\t"
       "out     0x3e, r29\n\t"
       "out     0x3f, r0 \n\t"
       "out     0x3d, r28\n\t"
```



```
"movw     r30, r24 \n\t"
"eor      r16, r16 \n\t"
"eor      r17, r17 \n\t"
"ldi      r18, 0x01\n\t"
"ldi      r19, 0x04\n\t"
"ldi      r20, 0x00\n\t"
"sub      r30, r18 \n\t"
"sbc      r31, r1  \n\t"

"add      r30, r18 \n\t"
"adc      r31, r1  \n\t"
"ld       r5 , Z   \n\t"
"mov      r9 , r5  \n\t"
"add      r9 , r9  \n\t" // plainText[0][i] << 1
"mov      r10, r5  \n\t"
"add      r10, r10 \n\t"
"eor      r10, r10 \n\t"
"adc      r10, r10 \n\t" // plainText[0][i] >> 7
"ldi      r21, 0x1b\n\t"
"mul      r10, r21 \n\t"
"eor      r9 , r0  \n\t" // (plainText[0][i] << 1) ^ ((plainText[0][i] >> 7) * 0x1b)

"ldd      r6 , Z+4 \n\t"
"eor      r9 , r6  \n\t"

"mov      r10, r6  \n\t"
"add      r10, r10 \n\t" // plainText[1][i] << 1
"mov      r11, r6  \n\t"
"add      r11, r11 \n\t"
"eor      r11, r11 \n\t"
"adc      r11, r11 \n\t" // plainText[1][i] >> 7
"mul      r11, r21 \n\t"
"eor      r10, r0  \n\t"

"ldd      r7 , Z+8 \n\t"
"eor      r10, r7  \n\t"
"ldd      r8 , Z+12\n\t"
"eor      r10, r8  \n\t"
"mov      r11, r9  \n\t"
"eor      r11, r10 \n\t" // temp[0][i]
```



```
"mov      r12, r7  \n\t"
"add      r12, r12 \n\t" // plainText[2][i] << 1
"mov      r13, r7  \n\t"
"add      r13, r13 \n\t"
"eor      r13, r13 \n\t"
"adc      r13, r13 \n\t" // plainText[2][i] >> 7
"mul      r13, r21 \n\t"
"eor      r12, r0  \n\t"
"eor      r12, r5  \n\t"
"eor      r10, r12 \n\t" // temp[1][i]
///////////////////////

"mov      r13, r8  \n\t"
"add      r13, r13 \n\t"
"mov      r14, r8  \n\t"
"add      r14, r14 \n\t"
"eor      r14, r14 \n\t"
"adc      r14, r14 \n\t"
"mul      r14, r21 \n\t"
"eor      r13, r0  \n\t"
"eor      r12, r13 \n\t"
"eor      r12, r8  \n\t"
"eor      r12, r6  \n\t"
///////////////////////

"eor      r13 , r9   \n\t"
"eor      r7  , r13  \n\t" // temp[3][i]
"eor      r7  , r5   \n\t"
///////////////////////

"st       Z    , r11 \n\t"
"std      Z+4  , r10 \n\t"
"std      Z+8  , r12 \n\t"
"std      Z+12,  r7  \n\t"

"add      r16, r18 \n\t"
"cp       r16, r19 \n\t"
"cpc      r17, r20 \n\t"
"brne     .-120 \n\t"
```



```
"adiw     r28, 0x10\n\t"
"in       r0 , 0x3f\n\t"
"cli  \n\t"

"out      0x3e, r29\n\t"
"out      0x3f, r0 \n\t"
"out      0x3d, r28\n\t"

"pop      r29 \n\t"
"pop      r28 \n\t"
"pop      r23 \n\t"
"pop      r22 \n\t"
"pop      r21 \n\t"
"pop      r20 \n\t"
"pop      r19 \n\t"
"pop      r18 \n\t"
"pop      r17 \n\t"
"pop      r16 \n\t"
"pop      r14 \n\t"
"pop      r13 \n\t"
"pop      r12 \n\t"
"pop      r11 \n\t"
"pop      r10 \n\t"
"pop      r9 \n\t"
"pop      r8 \n\t"
"pop      r7 \n\t"
"pop      r6 \n\t"
"pop      r5 \n\t"
```

Inline Assembly의 장점

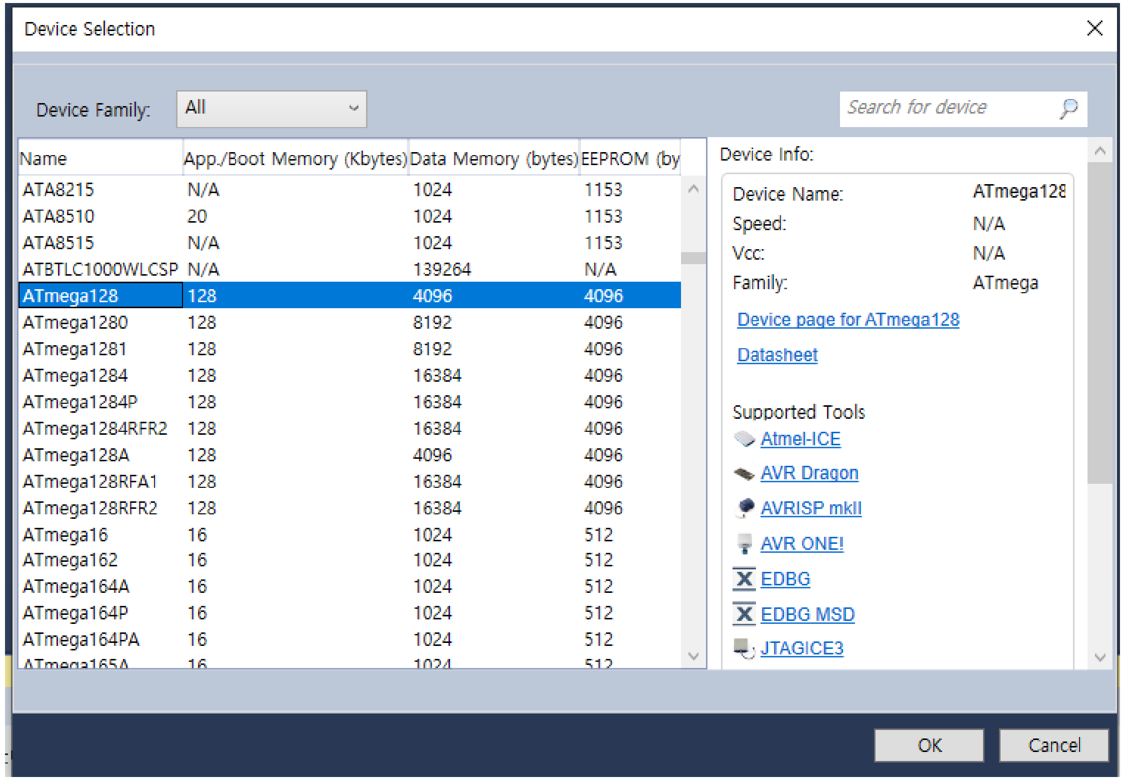
- C 언어 코드에 비해 코드 수를 상당히 줄일 수 있다.

  -> -O0 기준 mixcolumns 코드 수 = 1100

  inline Assembly 코드 수 = 250

- 최소 메모리로 최고의 사양을 낼 수 있다.

4. Atmel Studio를 이용한 디버깅



Device Selection

Device Family: All

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (by
ATA8215	N/A	1024	1153
ATA8510	20	1024	1153
ATA8515	N/A	1024	1153
ATBTLC1000WLCSP	N/A	139264	N/A
ATmega128	128	4096	4096
ATmega1280	128	8192	4096
ATmega1281	128	8192	4096
ATmega1284	128	16384	4096
ATmega1284P	128	16384	4096
ATmega1284RFR2	128	16384	4096
ATmega128A	128	4096	4096
ATmega128RFA1	128	16384	4096
ATmega128RFR2	128	16384	4096
ATmega16	16	1024	512
ATmega162	16	1024	512
ATmega164A	16	1024	512
ATmega164P	16	1024	512
ATmega164PA	16	1024	512
ATmega165A	16	1024	512

Device Info:

Device Name: ATmega128

Speed: N/A

Vcc: N/A

Family: ATmega

Device page for ATmega128

Datasheet

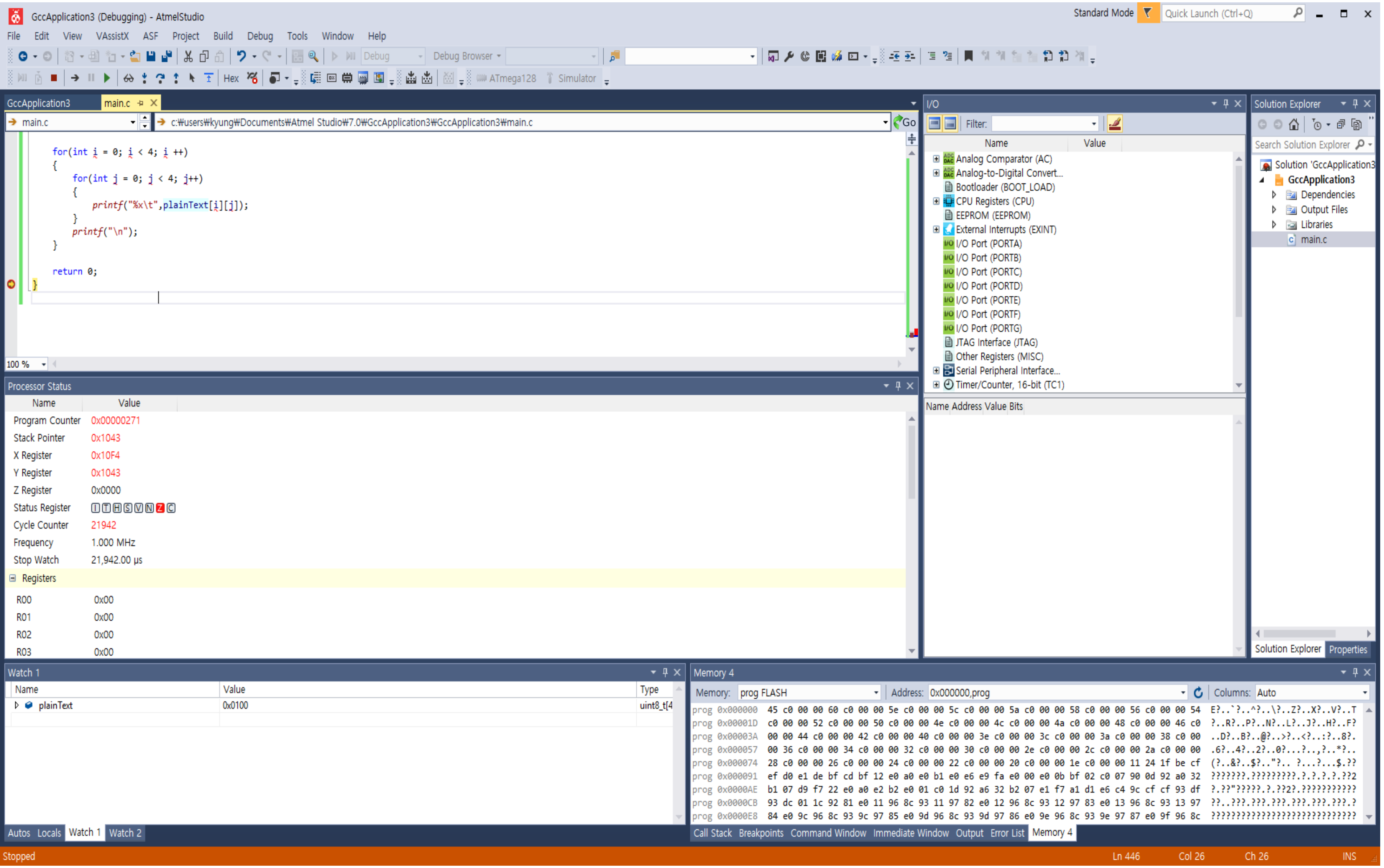
Supported Tools

- Atmel-ICE
- AVR Dragon
- AVRISP mkII
- AVR ONE!
- EDBG
- EDBG MSD
- JTAGICE3

IDE = Atmel Studio 7.0

타겟 보드 = Atmel 128

보드가 없기 때문에 Simulator로 진행


GccApplication3 (Debugging) - AtmelStudio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Debug Browser
ATmega128 Simulator
GccApplication3
main.c
I/O
Solution Explorer
Processor Status
Watch 1
Memory 4
Autos Locals Watch 1 Watch 2
Call Stack Breakpoints Command Window Immediate Window Output Error List Memory 4
Stopped
Ln 446 Col 26 Ch 26 INS

c:₩users₩kyung₩Documents₩Atmel Studio₩7.0₩GccApplication3₩GccApplication3₩main.c

```c
    for(int i = 0; i < 4; i ++)
    {
        for(int j = 0; j < 4; j++)
        {
            printf("%x\t",plainText[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Name	Value
Program Counter	0x00000271
Stack Pointer	0x1043
X Register	0x10F4
Y Register	0x1043
Z Register	0x0000
Status Register	ITHSVNZC
Cycle Counter	21942
Frequency	1.000 MHz
Stop Watch	21,942.00 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00

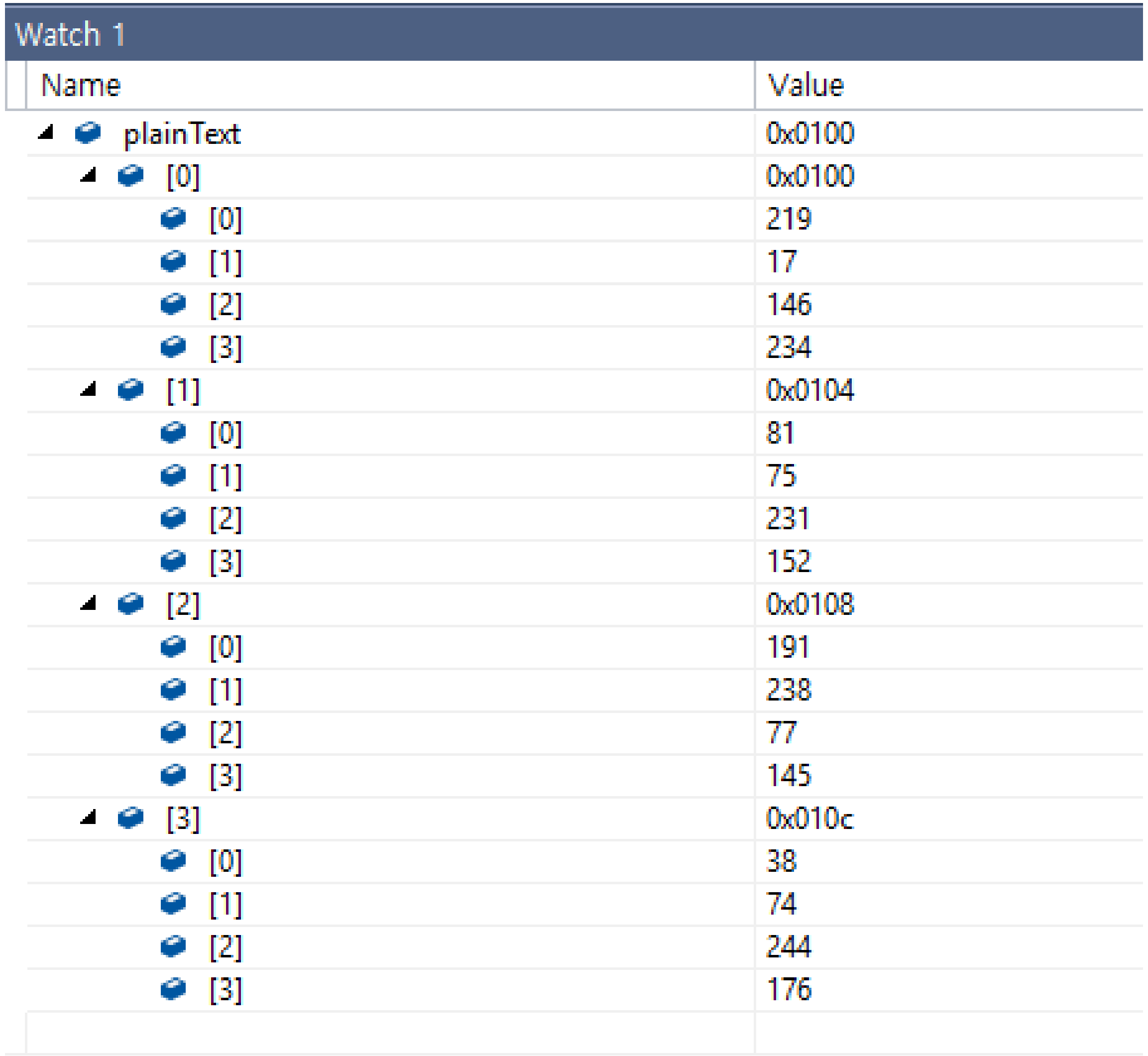
Name	Value	Type
plainText	0x0100	uint8_t[4

Memory: prog FLASH Address: 0x000000,prog Columns: Auto

```
prog 0x000000  45 c0 00 00 60 c0 00 00 5e c0 00 00 5c c0 00 00 5a c0 00 00 58 c0 00 00 56 c0 00 00 54  E?..`?..^?..\?..Z?..X?..V?..T
prog 0x00001D  c0 00 00 52 c0 00 00 50 c0 00 00 4e c0 00 00 4c c0 00 00 4a c0 00 00 48 c0 00 00 46 c0  ?..R?..P?..N?..L?..J?..H?..F?
prog 0x00003A  00 00 44 c0 00 00 42 c0 00 00 40 c0 00 00 3e c0 00 00 3c c0 00 00 3a c0 00 00 38 c0 00  ..D?..B?..@?..>?..<?..:?..8?.
prog 0x000057  00 36 c0 00 00 34 c0 00 00 32 c0 00 00 30 c0 00 00 2e c0 00 00 2c c0 00 00 2a c0 00 00  .6?..4?..2?..0?...?..,?..*?..
prog 0x000074  28 c0 00 00 26 c0 00 00 24 c0 00 00 22 c0 00 00 20 c0 00 00 1e c0 00 00 11 24 1f be cf  (?..&?..$?.."?.. ?...?...$.??
prog 0x000091  ef d0 e1 de bf cd bf 12 e0 a0 e0 b1 e0 e6 e9 fa e0 00 e0 0b bf 02 c0 07 90 0d 92 a0 32  ???????.????????.?.?.?.?.??2
prog 0x0000AE  b1 07 d9 f7 22 e0 a0 e2 b2 e0 01 c0 1d 92 a6 32 b2 07 e1 f7 a1 d1 e6 c4 9c cf cf 93 df  ?.??"?????.?.??2?.??????????
prog 0x0000CB  93 dc 01 1c 92 81 e0 11 96 8c 93 11 97 82 e0 12 96 8c 93 12 97 83 e0 13 96 8c 93 13 97  ??..???.???.???.???.???.???.?
prog 0x0000E8  84 e0 9c 96 8c 93 9c 97 85 e0 9d 96 8c 93 9d 97 86 e0 9e 96 8c 93 9e 97 87 e0 9f 96 8c  ?????????????????????????????
```

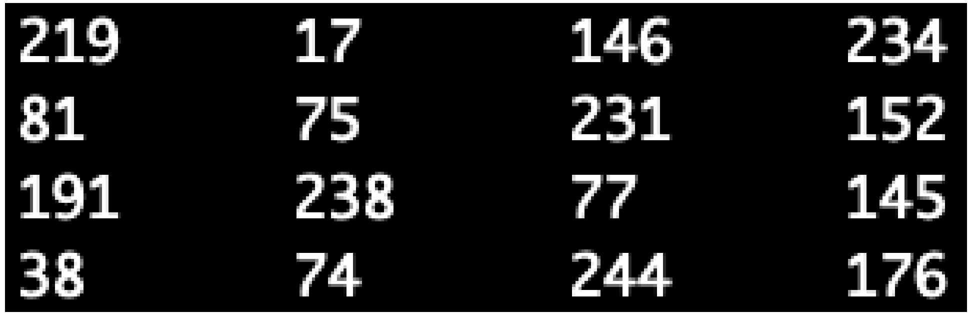
소스 코드 복사 후 BreakPoint 설정

BreakPoint에서 작업 중지 후 메모리 및 레지스터 값 확인



Watch 1

Name	Value
plainText	0x0100
[0]	0x0100
[0]	219
[1]	17
[2]	146
[3]	234
[1]	0x0104
[0]	81
[1]	75
[2]	231
[3]	152
[2]	0x0108
[0]	191
[1]	238
[2]	77
[3]	145
[3]	0x010c
[0]	38
[1]	74
[2]	244
[3]	176



```
219     17      146     234
81      75      231     152
191     238     77      145
38      74      244     176
```

Simulator 의 plainText값과 gcc로 컴파일 한 plainText 값이 동일한 것을 확인

감사합니다.