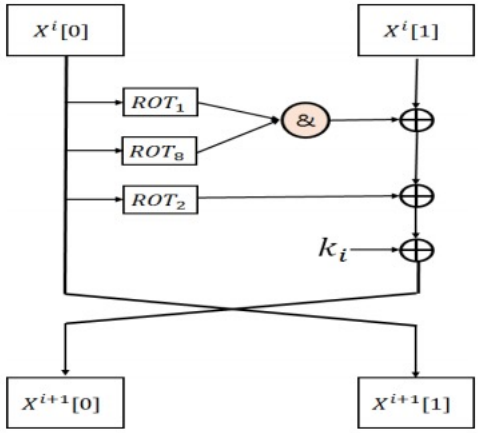
SIMON-SPECK
&
RISC-V

https://youtu.be/MbTFGbFLTbo

SIMON-SPECK

- SIMON
  - AND, Rotate, XOR을 사용하는 Feistel 구조

Block size($2n$)	Key size	Word size(n)	Round
32	64	16	32
48	72	24	36
	96		36
64	96	32	42
	128		44
96	96	48	52
	144		54
128	128	64	68
	192		69
	256		72



SIMON-SPECK

- SIMON Reference Code

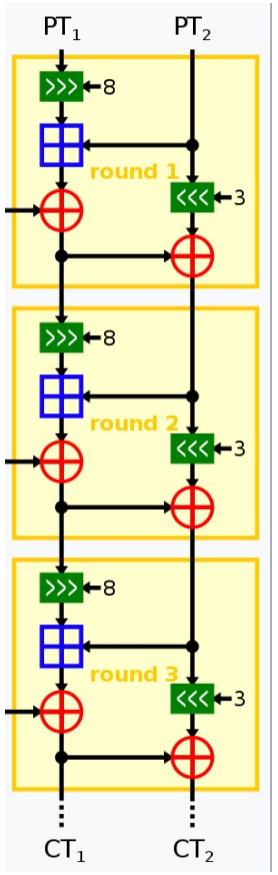
```
#define f32(x) ((ROTL32(x,1) & ROTL32(x,8)) ^ ROTL32(x,2))
#define R32x2(x,y,k1,k2) (y^=f32(x), y^=k1, x^=f32(y), x^=k2)


void Simon6496KeySchedule(u32 K[],u32 rk[])
{
  u32 i,c=0xfffffffc;
  u64 z=0x7369f885192c0ef5LL;

  rk[0]=K[0]; rk[1]=K[1]; rk[2]=K[2];

  for(i=3;i<42;i++){
    rk[i]=c^(z&1)^rk[i-3]^ROTR32(rk[i-1],3)^ROTR32(rk[i-1],4);
    z>>=1;
  }
}



void Simon6496Encrypt(u32 Pt[],u32 Ct[],u32 rk[])
{
  u32 i;

  Ct[1]=Pt[1]; Ct[0]=Pt[0];
  for(i=0;i<42;) R32x2(Ct[1],Ct[0],rk[i++],rk[i++]);
}
```

```
void Simon6496Enc(u32 pt[], u32 rk[]) {
    u32 i, temp;

    for(i=0; i<1; i+=2){
        temp = ROL32( x: pt[1],  n: 1);
        temp &= ROL32( x: pt[1],  n: 8);
        temp ^= ROL32( x: pt[1],  n: 2);
        pt[0] ^= temp;
        pt[0] ^= rk[i];

        temp = ROL32( x: pt[0],  n: 1);
        temp &= ROL32( x: pt[0],  n: 8);
        temp ^= ROL32( x: pt[0],  n: 2);
        pt[1] ^= temp;
        pt[1] ^= rk[i+1];
    }
}
```

SIMON-SPECK

- SPECK
  - Addition, Rotation, XOR을 사용하는 Feistel 구조

Block size (bits)	Key size (bits)	Rounds
2×16 = 32	4×16 = 64	22
2×24 = 48	3×24 = 72	22
	4×24 = 96	23
2×32 = 64	3×32 = 96	26
	4×32 = 128	27
2×48 = 96	2×48 = 96	28
	3×48 = 144	29
2×64 = 128	2×64 = 128	32
	3×64 = 192	33
	4×64 = 256	34



SIMON-SPECK

- SPECK Reference Code

```
#define ER32(x,y,k) (x=ROTR32(x,8), x+=y, x^=k, y=ROTL32(y,3), y^=x)
#define DR32(x,y,k) (y^=x, y=ROTR32(y,3), x^=k, x-=y, x=ROTL32(x,8))


void Speck6496KeySchedule(u32 K[],u32 rk[])
{
  u32 i,C=K[2],B=K[1],A=K[0];

  for(i=0;i<26;){
    rk[i]=A; ER32(B,A,i++);
    rk[i]=A; ER32(C,A,i++);
  }
}



void Speck6496Encrypt(u32 Pt[],u32 Ct[],u32 rk[])
{
  u32 i;

  Ct[0]=Pt[0]; Ct[1]=Pt[1];
  for(i=0;i<26;) ER32(Ct[1],Ct[0],rk[i++]);
}
```

```
void speck6496Enc(u32 *pt, u32 *rk){
    u32 i;
    for(i=0;i<26; i++) {
//          ER32(pt[1], pt[0], rk[i++]);
        pt[1] = ROR32( x: pt[1], n: 8);
        pt[1] += pt[0];
        pt[1] ^= rk[i];
        pt[0] = ROL32( x: pt[0], n: 3);
        pt[0] ^= pt[1];
    }
}
```

RISC-V

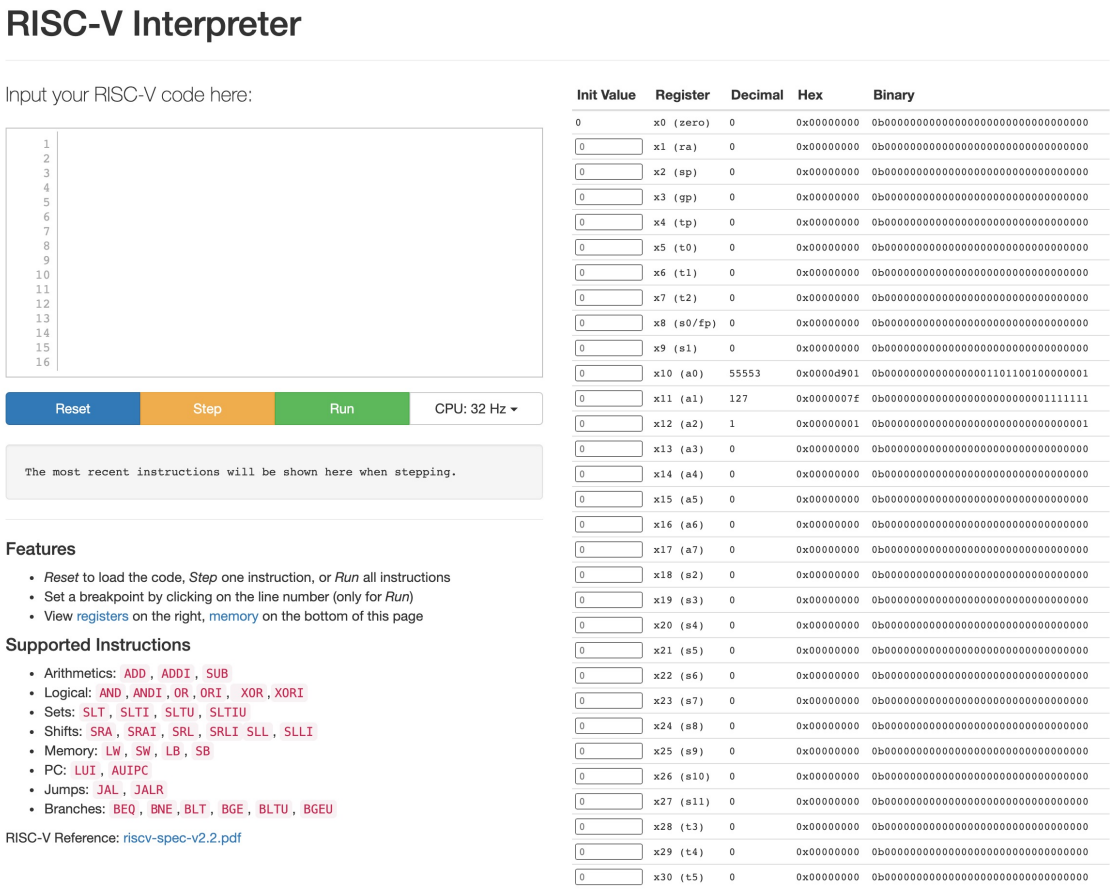
- 오픈 소스로 제공되는 명령어 셋을 기반으로한 프로세서

- 32-비트 구조인 RV32I 에서는 32개의 32-비트 레지스터를 제공

X0	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15
ZERO	RA	SP	GP	TP	T0	T1	T2	S0	S1	A0	A1	A2	A3	A4	A5
X16	X17	X18	X19	X20	X21	X22	X23	X24	X25	X26	X27	X28	X29	X30	X31
A6	A7	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	T3	T4	T5	T6

RISC-V

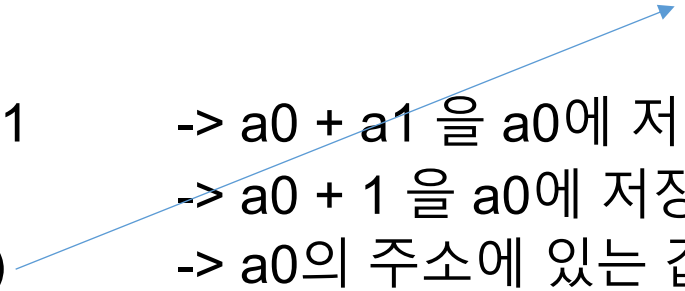
- https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/#



RISC-V

- 명령어 사용
  - OPCODE DEST, OP1, OP2

x[0]	x[1]	x[2]	x[3]
0	4	8	12



- ADD a0, a0, a1 -> a0 + a1 을 a0에 저장
- ADDI a0, a0, 1 -> a0 + 1 을 a0에 저장
- LW a1, 0(a0) -> a0의 주소에 있는 값을 a1에 저장
- SW a1, 0(a0) -> a1에 있는 값을 a0주소에 저장
- SLLI a0, a0, 1 -> 왼쪽 쉬프트 1번 값을 a0에 저장
- SRLI a0, a0, 1 -> 오른쪽 쉬프트 1번 한 값을 a0에 저장
- BNE a0, zero, loop -> a0와 zero가 같지 않다면 loop로 돌아감

RISC-V

- SIMON on risc-v
  - simon 64/96

```
simonEnc:
    /*
    데이터 불러오기
    a0 -> &pt
    a1 -> &rk
    */

    lw      a2, 0(a0)   //pt[0]
    lw      a3, 4(a0)   //pt[1]

    //loop (t3 = counter)
    li      t3, 21      //loop counter set 21(round 42)
    loop:
    macro_enc
    addi    a1, 8
    addi    t3, -1
    bne     t3, zero, loop
    //end loop

    sw      a3, 4(a0)   //pt[1]
    sw      a2, 0(a0)   //pt[0]
    ret
```

```
/*
    a1 : round key address
    a2 : pt[0]
    a3 : pt[1]
    a4 : rk[i]
    a5 : rk[i+1]
    t0, t1, a6, a7 : temp
*/
```

```
void Simon6496Enc(u32 pt[], u32 rk[]) {
    u32 i, temp;

    for(i=0; i<1; i+=2){
        temp = ROL32( x: pt[1],  n: 1);
        temp &= ROL32( x: pt[1],  n: 8);
        temp ^= ROL32( x: pt[1],  n: 2);
        pt[0] ^= temp;
        pt[0] ^= rk[i];

        temp = ROL32( x: pt[0],  n: 1);
        temp &= ROL32( x: pt[0],  n: 8);
        temp ^= ROL32( x: pt[0],  n: 2);
        pt[1] ^= temp;
        pt[1] ^= rk[i+1];
    }
}
```

```
.macro macro_enc rk
    lw      a4, 0+\rk
    lw      a5, 4+\rk

    //ROL(pt[1],1)
    slli    t0, a3, 1
    srli    t1, a3, 31
    or      a6, t0, t1

    //temp &= ROL(pt[1],8)
    slli    t0, a3, 8
    srli    t1, a3, 24
    or      a7, t0, t1

    and     a6, a6, a7

    //temp ^= ROL(pt[1], 2)
    slli    t0, a3, 2
    srli    t1, a3, 30
    or      a7, t0, t1

    xor     a6, a6, a7

    xor     a2, a2, a6
    xor     a2, a2, a4
```

```
    //ROL(pt[0],1)
    slli    t0, a2, 1
    srli    t1, a2, 31
    or      a6, t0, t1

    //temp &= ROL(pt[0],8)
    slli    t0, a2, 8
    srli    t1, a2, 24
    or      a7, t0, t1

    and     a6, a6, a7

    //temp ^= ROL(pt[0], 2)
    slli    t0, a2, 2
    srli    t1, a2, 30
    or      a7, t0, t1

    xor     a6, a6, a7

    xor     a3, a3, a6
    xor     a3, a3, a5
.endm
```

RISC-V

- SPECK on risc-v
  - speck 64/96

```
speck_enc:
    lw      a2, 0(a0)
    lw      a3, 4(a0)

    li      t0, 26

    loop:
        macro_enc
        addi    a1, 4
        addi    t0, -1
        bne     t0, zero, loop

    sw      a2, 0(a0)
    sw      a3, 4(a0)
    ret
```

```
/*
    a0 : plain text address
    a1 : roundkey text address
    a2 : pt[0]
    a3 : pt[1]
    a4 : rk[i]
    a5 : temp
    a6 : temp
    t0 : loop counter
*/
```

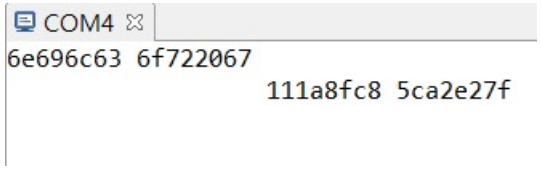
```
void speck6496Enc(u32 *pt, u32 *rk){
    u32 i;
    for(i=0;i<1; i++) {
//          ER32(pt[1], pt[0], rk[i++]);
        pt[1] = ROR32( x: pt[1],  n: 8);
        pt[1] += pt[0];
        pt[1] ^= rk[i];
        pt[0] = ROL32( x: pt[0],  n: 3);
        pt[0] ^= pt[1];
    }
}
```

```
.macro macro_enc
    lw      a4, 0(a1)

    //pt[1] = ROR32(pt[1], 8);
    srli    a3, a3, 8
    slli    a5, a3, 24
    or      a3, a3, a5

    //pt[1] += pt[0];
    add     a3, a3, a2

    //pt[1] ^= rk[i];
    xor     a3, a3, a4

    //pt[0] = ROL32(pt[0], 3);
    slli    a2, a2, 3
    srli    a5, a2, 29
    or      a2, a2, a5

    //pt[0] ^= pt[1];
    xor     a2, a2, a3
.endm
```

RISC-V

```
extern void simon_enc(uint32_t *pt, uint32_t *rk);

uint32_t pt[2] = {0x6e696c63, 0x6f722067};
//uint32_t mk[3] = {0x03020100, 0x0b0a0908, 0x13121110};
uint32_t rk[42] = {0x03020100, 0x0b0a0908, 0x13121110, 0xffae9dce,
                   0xc4facc91, 0xc83d1bb6, 0xb5d510ff, 0x36e2c07c,
                   0x72709043, 0x1343f40e, 0xea417e40, 0x9e635793,
                   0xa6965478, 0x8b052e75, 0x884c5f47, 0xd0e4e598,
                   0xe3e80363, 0x35f020e1, 0x1afa1c76, 0xbee71ed6,
                   0x763d4d2a, 0x0ca19efc, 0x0046cb1b, 0x59ce0704,
                   0x3dfb4191, 0xcbd9e8cc, 0xf3f75b6d, 0xa34520b7,
                   0xba7ae12d, 0x60e056a6, 0xf6a8d0f4, 0x943a89c1,
                   0xb4db50fe, 0x3481f018, 0xee1d573f, 0x4806d097,
                   0x56feb8ff, 0x0e529452, 0xd6d654a4, 0x7eb6e8dd,
                   0x8990d838, 0xb082bddc};

int main() {
    for(int i=0; i<2; i++) printf("%x ", pt[i]);
    printf("\n");
//  uint64_t oldcount = getcycles();

    simon_enc(pt, rk);

//    uint64_t cyclecount = getcycles()-oldcount;

    for(int i=0; i<2; i++) printf("%x ", pt[i]);

//    printf("cyc: %u", (unsigned int)cyclecount);
}
```

pt = 63 6c 69 6e 67 20 72 6f



Ct=(Ct[1],Ct[0])=(5ca2e27f,111a8fc8)

Q & A