(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2019년03월26일
(11) 등록번호 10-1962104
(24) 등록일자 2019년03월20일

(51) 국제특허분류(Int. Cl.)
G06F 9/30 (2018.01) *G06F 9/38* (2006.01)
H04L 9/08 (2006.01)
(21) 출원번호 10-2014-7014337
(22) 출원일자(국제) 2012년09월20일
심사청구일자 2017년08월25일
(85) 번역문제출일자 2014년05월28일
(65) 공개번호 10-2014-0093695
(43) 공개일자 2014년07월28일
(86) 국제출원번호 PCT/GB2012/052315
(87) 국제공개번호 WO 2013/072657
국제공개일자 2013년05월23일
(30) 우선권주장
1119834.8 2011년11월17일 영국(GB)
(56) 선행기술조사문헌
JP2004516706 A
KR1020150079731 A
KR1020150112782 A

(73) 특허권자
에이알엠 리미티드
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110
(72) 발명자
홀스넬 매튜 제임스
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드
그리센드와이트 리차드 로이
영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드
(뒷면에 계속)
(74) 대리인
이화익

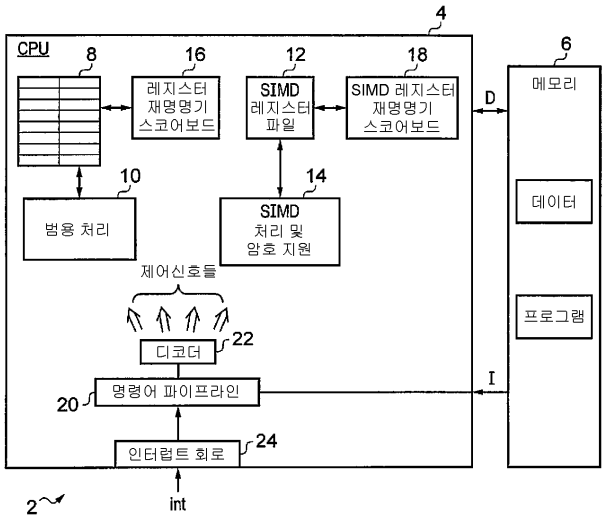
전체 청구항 수 : 총 27 항 심사관 : 유진태

(54) 발명의 명칭 **암호 알고리즘에서 해시값의 생성을 지원하는 SIMD 명령어**

(57) 요 약

데이터 처리 시스템(2)은, 단일 명령어 다중 데이터 레지스터 파일(12)과 단일 명령어 다중 처리회로(14)를 구비한다. 그 단일 명령어 다중 처리회로(14)는, 해시 알고리즘의 일부를 행하기 위한 암호 처리 명령어의 실행을 지원한다. 상기 단일 명령어 다중 데이터 레지스터 파일(12)내에는 오퍼랜드들이 기억되어 있다. 그 암호 지원 명령어는, 통상의 레인 기반 처리를 추종하지 않고, 상이한 출력 오퍼랜드의 부분들이 입력 오퍼랜드내의 다수의 상이한 요소에 의존하는 상기 출력 오퍼랜드를 생성한다.

대 표 도 - 도1



(72) 발명자

커셔 다니엘

영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드

빌레스 스튜어트 데이비드

영국 캠브리지 씨비1 9엔제이 체리힌톤 풀번로드 110 에이알엠 리미티드

명 세 서

청구범위

청구항 1

단일 명령어 다중 데이터 레지스터 파일; 및

상기 단일 명령어 다중 데이터 레지스터 파일에 연결되고, 단일 명령어 다중 데이터 프로그램 명령어에 의해 제어되어 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터내에 별도의 레인내에 기억된 별도의 데이터 요소에 관해서 독립적으로 처리 연산을 행하도록 구성된, 단일 명령어 다중 데이터 처리회로를 구비하고,

상기 단일 명령어 다중 데이터 처리회로는, 추가의 프로그램 명령어에 의해 제어되어, 상기 단일 명령어 다중 데이터 레지스터 파일의 출력 오퍼랜드용 레지스터내에 기억된 출력 오퍼랜드를 발생시키기 위해 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터내에 유지된 데이터 요소의 시퀀스를 포함하는 벡터 데이터 값에 관한 추가의 처리연산을 행하되, 상기 출력 오퍼랜드의 제1 부분이 상기 데이터 요소의 시퀀스내에 모든 데이터 요소에 의존한 값을 갖도록 구성된, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 2

제 1 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 벡터 데이터 값을 형성하는 복수의 워드의 데이터에 따라 출력 해시값을 상기 출력 오퍼랜드로서 생성하도록 연산하는 암호 프로그램 명령어인, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 3

제 2 항에 있어서,

상기 추가의 프로그램 명령어는, 연속적인 워드의 데이터와 중간 해시값의 적어도 일부를 소비하는 반복 처리 연산을 행하여 상기 출력 해시값을 생성하는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 4

제 1 항, 제 2 항 또는 제 3 항 중 어느 한 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 Vm[$32*2^{N}$-1: 0]를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 Qd_{output}[127:0]를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

```
X[127:0] = Qd[127:0];
Y[31:0] = Sn[31:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                t1[31:0] = OP FUNC (X[63:32], X[95:64], X[127:96]);

               Y[31:0] = Y[31:0] + ROL(X[31:0], 5) + T1[31:0] + Vm[Index+31:Index];
               X[63:32] = ROL(X[63:32], 30);
               T2[31:0] = Y[31:0];
               Y[31:0] = X[127:96];
               X[127:0] = {X[95:0]:T2[31:0]}
        }
Qd_output[127:0] = X[127:0];
```

여기서, OP FUNC(B,C,D)는

(((C XOR D) AND B)XOR D);

(B XOR C XOR D); 및

(B AND C) OR ((B OR C) AND D)중 하나이고,

ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

청구항 5

제 1 항, 제 2 항 또는 제 3 항 중 어느 한 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 $Vm[32*2^N-1: 0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 $Qd_{output}[127:0]$를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

```
X[127:0] = Qd[127:0];
Y[31:0] = Sn[31:0];
for (I = 0 to (2^N-1));
        {
                Index    = (I * 32);
                T1[31:0]  = OP_FUNC(X[63:32], X[95:64], X[127:96]);
                Y        = Y + ROL(X[31:0], 5) + T1[31:0] + Vm[(Index + 31):Index];
                X[63:32]  = ROL(X[63:32], 30);
                T2[31:0]  = Y;
                Y        = X[127:96];
                X[127:0]  = {X[95:0]:T2[31:0]};
        }
Qd_output[127:0] = {0:Y[31:0]};
```

여기서, OP FUNC(B,C,D)는,

(((C XOR D) AND B)XOR D);

(B XOR C XOR D); 및

(B AND C) OR ((B OR C) AND D)중 하나이고,

ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

청구항 6

제 4 항에 있어서,

상기 추가의 프로그램 명령어는, OP FUNC(B,C,D)로서

(((C XOR D) AND B)XOR D);

(B XOR C XOR D); 및

(B AND C) OR ((B OR C) AND D)중 하나를 선택하는 필드를 포함하는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 7

제 4 항에 있어서,

상기 제1 입력 오퍼랜드Qd[127:0]와 상기 제2 입력 오퍼랜드Sn[31:0]은, 상기 단일 명령어 다중 데이터 레지스터 파일내의 별도의 레지스터로부터 판독되는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 8

제 4 항에 있어서,

상기 제1 입력 오퍼랜드Qd[127:0]와 상기 제2 입력 오퍼랜드Sn[31:0]은, 상기 단일 명령어 다중 데이터 레지스터 파일내의 공유 레지스터로부터 판독되는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 9

제 1 항, 제 2 항 또는 제 3 항 중 어느 한 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 $Vm[32*2^{N}-1:0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 $Qd_{output}[127:0]$를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

```
X[127:0] = Qd[127:0];
Y[127:0] = Qn[127:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);
                TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);
                T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];
                X[127:96] = T1[31:0] + X[127:96];
                Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]
                T2[31:0] = Y[127:96];
                Y[127:0] = {Y[95:0]:X[127:96]};

                        X[127:0] = {X[95:0]:T2[31:0]}
        }
Qd_output[127:0] = X[127:0];
```

여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

청구항 10

제 1 항, 제 2 항 또는 제 3 항 중 어느 한 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 $Vm[32*2^{N}-1:0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 $Qd_{output}[127:0]$를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

```
X[127:0] = Qn[127:0];
Y[127:0] = Qd[127:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);
                TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);
                T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];
                X[127:96] = T1[31:0] + X[127:96];
                Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]
                T2[31:0] = Y[127:96];
                Y[127:0] = {Y[95:0]:X[127:96]};
                X[127:0] = {X[95:0]:T2[31:0]}
        }
Qd_output[127:0] = Y[127:0];
```

여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

청구항 11

제 9 항에 있어서,

상기 제1 입력 오퍼랜드Qd[127:0]와 상기 제2 입력 오퍼랜드Qn[127:0]은, 상기 단일 명령어 다중 데이터 레지스터 파일내의 별도의 레지스터로부터 판독되는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 12

제 9 항에 있어서,

상기 제1 입력 오퍼랜드Qd[127:0]와 상기 제2 입력 오퍼랜드Qn[127:0]은, 상기 단일 명령어 다중 데이터 레지스터 파일내의 공유 레지스터로부터 판독되는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 13

제 1 항, 제 2 항 또는 제 3 항 중 어느 한 항에 있어서,

상기 단일 명령어 다중 데이터 처리회로는, 상기 추가의 프로그램 명령어와 상기 단일 명령어 다중 데이터 프로그램 명령어의 처리를 관리하는 공통 메카니즘을 이용하는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 14

제 13 항에 있어서,

상기 처리를 관리하는 것은,

레지스터 재명명;

명령어 스케쥴링;

명령어 발행;

명령어 폐기; 및

명령어 인터럽트 중,

하나 이상을 포함하는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 15

제 4 항에 있어서,

상기 단일 명령어 다중 데이터 처리회로는, 입력 오퍼랜드Sm[31:0]를 갖고, 2개의 비트 위치만큼의 Sm[31:0]의 우회전으로 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sd[31:0]를 생성하는 회전 명령어에 의해 제어되도록 구성된, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 16

제 4 항에 있어서,

상기 단일 명령어 다중 데이터 처리회로는, 제1 입력 오퍼랜드Sp[127:0]와 제2 입력 오퍼랜드Sq[127:0]을 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sr[127:0]를 생성하는, 제1 스케쥴 갱신 명령어에 의해 제어되도록 구성된, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

T[127:0]={Sp[63:0]:Sq[127:64]} 및

Sr[127:0]=T[127:0] XOR Sr[127:0] XOR Sq[127:0].

청구항 17

제 14 항에 있어서,

상기 단일 명령어 다중 데이터 처리회로는, 입력 오퍼랜드Ss[127:0]를 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드St[127:0]를 생성하는, 제2 스케쥴 갱신 명령어에 의해 제어되도록 구성된, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

$$T[127:0] = St[127:0] \text{ XOR } \{32\{0\}:Ss[127:32]\};$$

$$St[95:0] = \{T[94:64]:T[95]:T[62:32]:T[63]:T[30:0]:T[31]\};$$ 및

$$St[127:96] = (\{T[126:96]:T[127]\}) \text{ XOR } (\{T[29:0]:T[31:30]\}).$$

청구항 18

제 9 항에 있어서,

상기 단일 명령어 다중 데이터 처리회로는, 입력 오퍼랜드Sp[127:0]를 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sq[127:0]를 생성하는, 제1 스케쥴 갱신 명령어에 의해 제어되도록 구성된, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

T[127:0] = {Sp[31:0]:Sq[127:32]};

T[127:0] = VecROR32(T[127:0], 7) XOR VecROR32(T[127:0], 18) XOR VecROR32(T[127:0], 3);

및

Sq[127:0] = VecADD32(T[127:0], Sq[127:0]),

여기서, VecROR32(A,B)는 A내에서 각 32비트 워드의 B비트 위치만큼의 별도의 우회전이고, VecADD32(A,B)는 A내에서 각 32비트 워드를 B내에서 대응한 32비트 워드에 별도로 가산하는 것이다.

청구항 19

제 18 항에 있어서,

상기 단일 명령어 다중 데이터 처리회로는, 제1 입력 오퍼랜드Sp[127:0]와 제2 입력 오퍼랜드Sq[127:0]를 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sr[127:0]를 생성하는, 제2 스케쥴 갱신 명령어에 의해 제어되도록 구성된, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치:

T0[127:0] = {Sq[31:0]:Sp[127:32]};

T1[63:0] = Sq[127:64];

T1[63:0] = VecROR32(T1[63:0], 17) XOR VecROR32(T1[63:0], 19) XOR VecROR32(T1[63:0], 10);

T3[63:0] = VecADD32(Sr[63:0], T0[63:0]);

T1[63:0] = VecADD32(T3[63:0], T1[63:0]);

T2[63:0] = VecROR32(T1[63:0], 17) XOR VecROR32(T1[63:0], 19) XOR VecROR32(T1[63:0], 10);

T3[63:0] = VecADD32(Sr[127:64], T0[127:64]);

및

Sr[127:0] = {VecADD32(T3[63:0], T2[63:0]):T1[63:0]},

여기서, VecROR32(A,B)는 A내에서 각 32비트 워드의 B비트 위치만큼의 별도의 우회전이고, VecADD32(A,B)는 A내에서 각 32비트 워드를 B내에서 대응한 32비트 워드에 별도로 가산하는 것이다.

청구항 20

제 1 항, 제 2 항 또는 제 3 항 중 어느 한 항에 있어서,

상기 단일 명령어 다중 데이터 레지스터 파일로부터 분리되어 있고, 상기 단일 명령어 다중 데이터 레지스터 파일내의 레지스터들보다 비트폭이 낮은 범용 레지스터들을 갖는, 범용 레지스터 파일과, 상기 범용 레지스터 파일에 연결되고, 범용 처리 명령어에 의해 제어되어 상기 범용 레지스터들 중 하나에 기억된 입력 오퍼랜드에 관한 처리 연산을 행하도록 구성된 범용 처리회로를 더 구비한, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 21

단일 명령어 다중 데이터 오퍼랜드를 기억하는 단일 명령어 다중 데이터 레지스터 파일 수단; 및

단일 명령어 다중 데이터 프로그램 명령어의 제어하에서 처리연산을 행하고, 상기 단일 명령어 다중 데이터 레지스터 파일 수단에 연결되고, 상기 처리연산이 상기 단일 명령어 다중 데이터 레지스터 파일 수단의 입력 오퍼랜드용 레지스터내에 별도의 레인내에 기억된 별도의 데이터 요소에 관해서 독립적으로 행해지는, 단일 명령어

다중 데이터 처리수단을 구비하고,

상기 단일 명령어 다중 데이터 처리수단은, 추가의 프로그램 명령어에 의해 제어되어, 상기 단일 명령어 다중 데이터 레지스터 파일 수단의 출력 오퍼랜드용 레지스터내에 기억된 출력 오퍼랜드를 발생시키기 위해 상기 단일 명령어 다중 데이터 레지스터 파일 수단의 입력 오퍼랜드용 레지스터내에 유지된 데이터 요소의 시퀀스를 포함하는 벡터 데이터 값에 관한 추가의 처리연산을 행하되, 상기 출력 오퍼랜드의 제1 부분이 상기 데이터 요소의 시퀀스내에 모든 데이터 요소에 의존한 값을 갖는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치.

청구항 22

단일 명령어 다중 데이터 오퍼랜드를 단일 명령어 다중 데이터 레지스터 파일에 기억시키는 단계;

단일 명령어 다중 데이터 프로그램 명령어의 제어하에서 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터내에 별도의 레인내에 기억된 별도의 데이터 요소에 관해서 독립적으로 처리연산을 행하는 단계; 및

추가의 프로그램 명령어의 제어하에서, 상기 단일 명령어 다중 데이터 레지스터 파일의 출력 오퍼랜드용 레지스터내에 기억된 출력 오퍼랜드를 발생시키기 위해 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터내에 유지된 데이터 요소의 시퀀스를 포함하는 벡터 데이터 값에 관한 추가의 처리연산을 행하는 단계로서, 상기 출력 오퍼랜드의 제1 부분이 상기 데이터 요소의 시퀀스내에 모든 데이터 요소에 의존한 값을 갖는, 단계를 포함한, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리방법.

청구항 23

제 22 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 $Vm[32*2^{N}-1: 0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 $Qd_{output}[127:0]$를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리방법:

$$X[127:0] = Qd[127:0];$$

Y[31:0] = Sn[31:0];
for (I = 0 to (2^{N}-1));
{
Index = (I*32);
t1[31:0] = OP FUNC (X[63:32], X[95:64], X[127:96]);
Y[31:0] = Y[31:0] + ROL(X[31:0], 5) + T1[31:0] + Vm[Index+31:Index];
X[63:32] = ROL(X[63:32], 30);
T2[31:0] = Y[31:0];
Y[31:0] = X[127:96];
X[127:0] = {X[95:0]:T2[31:0]}
}
Qd_{output}[127:0] = X[127:0];

여기서, OP FUNC(B,C,D)는,

(((C XOR D) AND B)XOR D);

(B XOR C XOR D); 및

(B AND C) OR ((B OR C) AND D)중 하나이고,

ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

청구항 24

제 22 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 $Vm[32*2^{N}-1: 0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 $Qd_{output}[127:0]$를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리방법:

X[127:0] = Qd[127:0];
Y[31:0] = Sn[31:0];
for (I = 0 to (2^{N}-1));
{
Index = (I * 32);
T1[31:0] = OP_FUNC(X[63:32], X[95:64], X[127:96]);
Y = Y + ROL(X[31:0], 5) + T1[31:0] + Vm[(Index + 31):Index];
X[63:32] = ROL(X[63:32], 30);
T2[31:0] = Y;
Y = X[127:96];
X[127:0] = {X[95:0]:T2[31:0]};
}

$$Qd_{output}[127:0] = \{0:Y[31:0]\};$$

여기서, OP FUNC(B,C,D)는,

(((C XOR D) AND B)XOR D);

(B XOR C XOR D); 및

(B AND C) OR ((B OR C) AND D)중 하나이고,

ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

청구항 25

제 22 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 $Vm[32*2^{N}-1: 0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 $Qd_{output}[127:0]$를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리방법:

X[127:0] = Qd[127:0];

Y[127:0] = Qn[127:0];

for (I = 0 to (2^N-1));

{

Index = (I*32);

TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);

TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);

T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];

X[127:96] = T1[31:0] + X[127:96];

Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]

T2[31:0] = Y[127:96];

Y[127:0] = {Y[95:0]:X[127:96]};

X[127:0] = {X[95:0]:T2[31:0]}

}

Qd_{output}[127:0] = X[127:0];

여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

청구항 26

제 22 항에 있어서,

상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드 Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 Vm[32*2^N-1:0]를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드 Qd_{output}[127:0]를 발생시키는, 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리방법:

X[127:0] = Qn[127:0];

Y[127:0] = Qd[127:0];

for (I = 0 to (2^N-1));

{

Index = (I*32);

TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);

TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);

T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];

X[127:96] = T1[31:0] + X[127:96];

Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]

T2[31:0] = Y[127:96];

Y[127:0] = {Y[95:0]:X[127:96]};

X[127:0] = {X[95:0]:T2[31:0]}

}

Qd_{output}[127:0] = Y[127:0];

여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

청구항 27

청구항 1, 2 또는 3 중 어느 한 항에 기재된 것과 같은 암호 알고리즘에서 해시값의 생성을 위한 데이터 처리장치에 대응한 가상 머신 실행 환경을 제공하도록 컴퓨터를 제어하는 컴퓨터 프로그램이 기억된 컴퓨터 판독 가능한 기억매체.

발명의 설명

기 술 분 야

[0001] 본 발명은, 데이터 처리 시스템 분야에 관한 것이다. 보다 구체적으로, 본 발명은, 데이터 처리 시스템 내에서 암호 지원 명령어를 제공하는 것에 관한 것이다.

배 경 기 술

[0002] 데이터 처리 시스템을 사용하여 암호 연산을 행하는 것은 공지되어 있다. 이러한 공지된 암호처리 연산의 예들로서는, 안전한 해시 알고리즘(SHA)이 있다. 그 SHA의 형태는 SHA-1, SHA-2, SHA256 및 SHA512를 포함하는 다양하고 상이한 것이 공지되어 있다. 이들 알고리즘은 계산 집약적이다.

[0003] 이들의 알고리즘을 지원하는 공지된 해결방법의 하나는, 범용 레지스터 파일을 갖는 범용 명령어를 실행하는 범용 프로세서를 사용하는 방법이다. 이 해결방법이 갖는 문제점은, 일반적으로 160비트 이상의 해시값을 생성할 수 있는 이들 알고리즘을 행할 때 연산되어야 하는 대량의 상태 데이터가, 한번에 상기 데이터의 일부를 연산하는 긴 연속적 개별 프로그램 명령어에 의해 상기 연산을 분할하여 행해야 하는 경우도 있어서, 그 알고리즘을 실행하는데 필요한 많은 시간과 그 알고리즘을 실행할 때 소비된 에너지가 불리하게 증가하게 되는 결과를 갖는다는 것이다.

[0004] 또 다른 해결방법으로는, 상기 알고리즘을 행하는 전용 회로를 갖고, 포인터가 상기 데이터가 해시되는 시작부를 지나간 후 그 결과의 해시값을 수신하기를 기다림으로써 시작되는 것이 일반적인, 전용 암호지원 프로세서, 이를테면 암호 코프로세서를 제공하는 방법이 있다. 이러한 해결방법이 갖는 문제점은, 상기 전용 암호 하드웨어의 제공으로 별도의 비용과 복잡함을 초래한다는 것이다. 또한, 상기 전용 하드웨어의 연산과, 인터럽트 핸들링이나 멀티태스킹 등의 상기 디바이스의 다른 연산을 통합할 때 문제가 생기는데, 그 이유는, 상기 전용 암호 하드웨어는, 데이터 처리 시스템내에 통상 구비된 메카니즘에 내장되어 상기 데이터 처리 시스템에 의한 상기 연산의 국면들을 처리하기 어렵고 복잡하기 때문이다.

발명의 내용

[0005] 일 국면에서 본 본 발명의 데이터 처리장치는,

[0006] 단일 명령어 다중 데이터 레지스터 파일; 및

[0007] 상기 단일 명령어 다중 데이터 레지스터 파일에 연결되고, 단일 명령어 다중 데이터 프로그램 명령어에 의해 제어되어 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터내에 별도의 레인(lane)내에 기억된 별도의 데이터 요소에 관해서 독립적으로 처리 연산을 행하도록 구성된, 단일 명령어 다중 데이터 처리회로를 구비하고,

[0008] 상기 단일 명령어 다중 데이터 처리회로는, 추가의 프로그램 명령어에 의해 제어되어, 내부에 기억된 출력 오퍼랜드를 발생시키기 위해 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터와 상기 단일 명령어 다중 데이터 레지스터 파일의 출력 오퍼랜드용 레지스터내에 유지된 데이터 요소의 시퀀스를 포함하는 벡터 데이터 값에 관한 추가의 처리연산을 행하되, 상기 출력 오퍼랜드의 제1 부분이 상기 데이터 요소의 시퀀스내에 모든 데이터 요소에 의존한 값을 갖도록 구성된다.

[0009] 본 기술에 의해 안 것은, 많은 데이터 처리 시스템에는 이미 단일 명령어 다중 데이터 처리 메카니즘이

구비되어 있다는 것이다. 이 단일 명령어 다중 데이터 처리 메카니즘은, 단일 명령어 다중 데이터 처리에서 일반적으로 포함된 큰 데이터 폭 오퍼랜드를 기억 및 조작 가능한 기억용량이 큰 단일 명령어 다중 데이터 레지스터 파일을 포함하는 것이 일반적이다. 단일 명령어 다중 데이터 처리에 있어서 별도의 레인의 데이터는 단일 프로그램 명령어의 제어하에서 독립적으로 처리되는 것이 보통이다. 예를 들면, 별도의 레인의 데이터는, 색상 화소값의 성분값, 또는 다른 벡터값, 스케일링등의 같은 처리 연산이 실시되는 모든 것을 포함하여도 된다. 본 기술에 의해 안 것은, 일반적인 형태의 단일 명령어 다중 데이터 프로그램 명령어를 추종하지 않는 추가의 프로그램 명령어로 단일 명령어 다중 데이터 레지스터 파일의 기억용량을 재사용할 수 있다는 것이다. 특히, 상기 레인의 처리는 독립적일 필요는 없고, 생성된 출력 오퍼랜드의 제1 부분은 입력을 형성하는 벡터 데이터 값내에서 데이터 요소 모두에 의존하는 값을 가질 수도 있다.

[0010] 단일 명령어 다중 데이터 프로그램 명령어의 영역 외측의 단일 명령어 다중 데이터 레지스터 파일의 재사용은, 데이터 압축 및 데이터 암호화 등의 다양한 영역에 적용되어도 된다. 특히, 상기 기술은, 데이터 암호화에 아주 적합하다.

[0011] 이와 관련해서, 상기 추가의 프로그램 명령어는, 연속적인 워드의 데이터와 중간 해시값의 적어부 일부를 소비하는 반복 처리 연산을 행하도록 구성되어 출력 해시값을 생성하여도 된다. 해시값 생성은, 매우 긴 오퍼랜드값들을 기억 및 연산할 능력을 갖는 대량의 데이터와 레지스터 파일의 연산을 필요로 하는 것이 일반적이다.

[0012] 일 형태의 추가의 프로그램 명령어는, 상기 추가의 프로그램 명령어가 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0] 양쪽을 갖고, 상기 벡터 데이터 값은 N이 양의 정수인 Vm[32*2^N-1: 0]를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드Qd_{output}[127:0]를 발생시킨다:

[0013]

```
X[127:0] = Qd[127:0];
Y[31:0] = Sn[31:0];
for (I = 0 to (2^N-1));
{
        Index = (I*32);
        t1[31:0] = OP FUNC (X[63:32], X[95:64], X[127:96]);
        Y[31:0] = Y[31:0] + ROL(X[31:1], 5) + T1[31:0] + Vm[Index+31:Index];
        X[63:32] = ROL(X[63:32], 30);
        T2[31:0] = Y[31:0];
        Y[31:0] = X[127:96];
        X[127:0] = {X[95:0]:T2[31:0]}
}
Qd_output[127:0] = X[127:0];
```

[0014]

[0015] 여기서, OP FUNC(B,C,D)는,

[0016] (((C XOR D) AND B)XOR D);

[0017] (B XOR C XOR D); 및

[0018] (B AND C) OR ((B OR C) AND D)중 하나이고,

[0019] ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

[0020] 이러한 형태의 반복 프로그램 명령어는, SHA-1 알고리즘을 구현하는데 매우 적합하다. 상기 규정된 연

산은 의사코드의 형태로 주어지고, 본 기술분야에서 당업자라면 알 수 있듯이 다양한 상이한 하드웨어 형태로 구현되어도 된다는 것을 알 것이다. 특히, 로우(low) 회로 오버헤드 구현은, 보다 고성능의 구현으로서 적어도 일부의 상이한 반복을 동시에 행하려고 하는 반복 연산을 형성하기 위해 값들을 재순환시킬 수도 있다.

[0021] 또 다른 형태의 상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0] 양쪽을 갖고, 상기 벡터 데이터값은 N이 양의 정수인 $Vm[32*2^{N}-1:0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드$Qd_{output}[127:0]$를 발생시킨다:

```
X[127:0] = Qd[127:0];
Y[31:0] = Sn[31:0];
for (I = 0 to (2^N-1));
        {
                Index    = (I * 32);
                T1[31:0]  = OP_FUNC(X[63:32], X[95:64], X[127:96]);
                Y        = Y + ROL(X[31:0], 5) + T1[31:0] + Vm[(Index + 31):Index];
                X[63:32]  = ROL(X[63:32], 30);
                T2[31:0]  = Y;
```

[0022]

```
                        Y        = X[127:96];
                        X[127:0]  = {X[95:0]:T2[31:0]};
        }
Qd_output[127:0] = {0:Y[31:0]};
```

[0023]

[0024] 여기서, OP FUNC(B,C,D)는,

[0025] (((C XOR D) AND B)XOR D);

[0026] (B XOR C XOR D); 및

[0027] (B AND C) OR ((B OR C) AND D)중 하나이고,

[0028] ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

[0029] OP FUNC에서 평가한 함수는, 상기 추가의 프로그램 명령어내에 특별한 필드에 따라 선택되어도 되거나, 해시되는 현재 입력된 블록의 데이터 값의 처리동안에 얼마나 많은 반복을 행하였는지에 따라 선택되어도 된다.

[0030] 일부의 실시예에서, 상기 단일 명령어 다중 데이터 레지스터 파일은, 상기 제1 입력 오퍼랜드와 상기 제2 입력 오퍼랜드 모두를 단일 레지스터에 기억할 수 없을 수도 있어서, 이들은 상기 단일 명령어 다중 데이터 레지스터 파일내의 별도의 레지스터내에 저장될 수도 있다. 그 밖의 실시예에서, 상기 제1 입력 오퍼랜드와 상기 제2 입력 오퍼랜드는, 공유 레지스터내에 저장되어도 되고 단일 입력 오퍼랜드로서 생각되어도 된다.

[0031] 또 다른 실시예에서, 상기 추가의 프로그램 명령어와 조합하거나 상기 추가의 프로그램 명령어 대신에, 본 기술은, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0] 양쪽을 갖는 상기 추가의 프로그램 명령어에 대해 지원하고, 상기 벡터 데이터값은 N이 양의 정수인 $Vm[32*2^{N}-1:0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드$Qd_{output}[127:0]$를 발생시킨다:

```
X[127:0] = Qd[127:0];
Y[127:0] = Qn[127:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);
                TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);
                T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];
                X[127:96] = T1[31:0] + X[127:96];
                Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]
```

[0032]

```
                T2[31:0] = Y[127:96];
                Y[127:0] = {Y[95:0]:X[127:96]};
                X[127:0] = {X[95:0]:T2[31:0]}
        }
Qd_output[127:0] = X[127:0];
```

[0033]

[0034] 여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

[0035] 마찬가지의 방식으로, 상기 추가의 프로그램 명령어는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0] 양쪽을 갖는 형태도 가져도 되고, 상기 벡터 데이터값은 N이 양의 정수인 $Vm[32*2^N-1: 0]$를 포함하고, 상기 추가의 처리 연산은 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드$Qd_{output}[127:0]$를 발생시킨다:

```
X[127:0] = Qn[127:0];
Y[127:0] = Qd[127:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);
                TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);
                T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];
                X[127:96] = T1[31:0] + X[127:96];
                Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]
                T2[31:0] = Y[127:96];
                Y[127:0] = {Y[95:0]:X[127:96]};
                X[127:0] = {X[95:0]:T2[31:0]}
        }
Qd_output[127:0] = Y[127:0];
```

[0036]

[0037] 여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND

D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

[0038] 상기 2개의 형태의 추가의 프로그램 명령어는, SHA-224 알고리즘과 SHA-256 알고리즘을 지원하는데 매우 적합하다.

[0039] 상기 추가의 프로그램 명령어의 처리를 관리하는 메카니즘(들)은, 편리하게 상기 단일 명령어 다중 데이터 처리회로와 조합되어도 된다. 상기 추가의 처리 명령어의 처리를 관리하는 메카니즘(들)은, 상기 단일 명령어 다중 데이터 명령어 레지스터 파일을 사용하고, 상기 추가의 프로그램 명령어(예를 들면, 인터럽트 핸들링, 스케쥴링)의 처리를 관리하는 메카니즘들이 상기 단일 명령어 다중 데이터 처리회로의 것과 통합될 때 구현이 단순화될 수 있다.

[0040] 상기 단일 명령어 다중 데이터 처리회로의 것과 통합되어도 되는 상기 추가의 프로그램 명령어의 처리를 관리하는 국면들은, 레지스터 재명명, 명령어 스케쥴링, 명령어 발행, 명령어 폐기 및 명령어 인터럽트를 포함한다. 상기 단일 명령어 다중 데이터 처리회로는, 이미 이들 연산을 관리 및 지원하는 회로요소를 구비하는 것이 일반적이고, 상기 추가의 프로그램 명령어는 이러한 관리 지원에 상대적으로 용이하게 통합되어도 된다. 이것은, 인터럽트가 암호 해시값의 생성에 의해 일부 일어나면, 통상의 인터럽트 핸들링 메카니즘들은 그 인터럽트를 서비스하고, 그 해시 계산을 추가의 비용이나 복잡함이 거의 없이 상기 인터럽트를 서비스한 후에 재시작하거나 또는 계속하는데 사용되어도 된다는 이점을 제공한다.

[0041] 해시 알고리즘의 지원은, 입력 오퍼랜드Sm[31:0]를 갖고 출력 오퍼랜드Sd[31:0]를 생성하는 회전 명령어에, 2개의 비트 위치만큼의 Sm[31:0]의 우회전으로 나타낸 것과 같은 값을 제공하는 것에 의해 더욱 개선된다.

[0042] 상기 중간 해시값의 생성과 추가에 의해 행해지는 암호 해시 알고리즘의 처리의 다른 국면은, 처리중인 상기 파일내에 데이터 요소들의 스케쥴의 갱신이다. 이러한 스케쥴의 갱신은, 그 해시 생성에 의한 작업 부하의 관점에서 균형이 맞추어져 처리 스루풋에서 불리한 병목현상이 생기지 않아야 한다. 이에 따라, 본 발명의 일부의 실시예는, 상기 단일 명령어 다중 데이터 처리회로가, 제1 입력 오퍼랜드Sp[127:0]와 제2 입력 오퍼랜드Sq[127:0]을 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sr[127:0]를 생성하는, 제1 스케쥴 갱신 명령어에 의해 제어되도록 구성된 것을 제공한다:

[0043] T[127:0]={Sp[63:0]:Sq[127:64]} 및

[0044] Sr[127:0]=T[127:0] XOR Sr[127:0] XOR Sq[127:0].

[0045] 또한, 일부의 실시예는, 상기 단일 명령어 다중 데이터 처리회로가, 입력 오퍼랜드Ss[127:0]를 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드St[127:0]를 생성하는, 제2 스케쥴 갱신 명령어에 의해 제어되도록 구성된 것을 제공한다:

[0046]
$$T[127:0] = St[127:0] \text{ XOR } \{32\{0\}:Ss[127:32]\};$$
$$St[95:0] = \{T[94:64]:T[95]:T[62:32]:T[63]:T[30:0]:T[31]\}; \text{ and}$$
$$St[127:96] = (\{T[126:96]:T[127]\}) \text{ XOR } (\{T[29:0]:T[31:30]\}).$$

[0047] 위의 2개의 형태의 프로그램 명령어는, 상기 SHA-256 및 상기 SHA-224 알고리즘을 지원하는데에 매우 적합하다.

[0048] 다른 형태의 해시 알고리즘에서 상기 스케쥴 생성의 지원을 돕기 위해서, 일부의 실시예는, 상기 단일 명령어 다중 데이터 처리회로가, 입력 오퍼랜드Sp[127:0]를 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sq[127:0]를 생성하는, 제1 스케쥴 갱신 명령어에 의해 제어되도록 구성된 것을 제공한다:

[0049]
$$T[127:0] = \{Sp[31:0]:Sq[127:32]\};$$
$$T[127:0] = VecROR32(T[127:0], 7) \text{ XOR } VecROR32(T[127:0], 18) \text{ XOR } VecROR32(T[127:0], 3);$$
$$\text{and}$$
$$Sq[127:0] = VecADD32(T[127:0], Sq[127:0]),$$

[0050] 여기서, VecROR32(A,B)는 A내에서 각 32비트 워드의 B비트 위치만큼의 별도의 우회전이고, VecADD32(A,B)는 A내에서 각 32비트 워드를 B내에서 대응한 32비트 워드에 별도로 가산하는 것이다.

[0051] 또 다른 실시예에서는, 상기 단일 명령어 다중 데이터 처리회로가, 제1 입력 오퍼랜드Sp[127:0]와 제2 입력 오퍼랜드Sq[127:0]를 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sr[127:0]를 생성하는, 제2 스케줄 갱신 명령어에 의해 제어되도록 구성된 것을 추가로 제공한다:

T0[127:0] = {Sq[31:0]:Sp[127:32]};

T1[63:0] = Sq[127:64];

T1[63:0] = VecROR32(T1[63:0], 17) XOR VecROR32(T1[63:0], 19) XOR VecROR32(T1[63:0], 10);

T3[63:0] = VecADD32(Sr[63:0], T0[63:0]);

T1[63:0] = VecADD32(T3[63:0], T1[63:0]);

T2[63:0] = VecROR32(T1[63:0], 17) XOR VecROR32(T1[63:0], 19) XOR VecROR32(T1[63:0], 10);

T3[63:0] = VecADD32(Sr[127:64], T0[127:64]); and

[0052] Sr[127:0] = {VecADD32(T3[63:0], T2[63:0]):T1[63:0]},

[0053] 여기서, VecROR32(A,B)는 A내에서 각 32비트 워드의 B비트 위치만큼의 별도의 우회전이고, VecADD32(A,B)는 A내에서 각 32비트 워드를 B내에서 대응한 32비트 워드에 별도로 가산하는 것이다.

[0054] 위의 2개의 형태의 프로그램 명령어는, 상기 SHA-256 알고리즘을 지원하는데에 매우 적합하다.

[0055] 다른 국면에서 본 본 발명의 데이터 처리장치는,

[0056] 단일 명령어 다중 데이터 오퍼랜드를 기억하는 단일 명령어 다중 데이터 레지스터 파일 수단; 및

[0057] 단일 명령어 다중 데이터 프로그램 명령어의 제어하에서 처리연산을 행하고, 상기 단일 명령어 다중 데이터 레지스터 파일 수단에 연결되고, 상기 처리연산이 상기 단일 명령어 다중 데이터 레지스터 파일 수단의 입력 오퍼랜드용 레지스터내에 별도의 레인내에 기억된 별도의 데이터 요소에 관해서 독립적으로 행해지는, 단일 명령어 다중 데이터 처리수단을 구비하고,

[0058] 상기 단일 명령어 다중 데이터 처리수단은, 추가의 프로그램 명령어에 의해 제어되어, 내부에 기억된 출력 오퍼랜드를 발생시키기 위해 상기 단일 명령어 다중 데이터 레지스터 파일 수단의 입력 오퍼랜드용 레지스터와 상기 단일 명령어 다중 데이터 레지스터 파일 수단의 출력 오퍼랜드용 레지스터내에 유지된 데이터 요소의 시퀀스를 포함하는 벡터 데이터 값에 관한 추가의 처리연산을 행하되, 상기 출력 오퍼랜드의 제1 부분이 상기 데이터 요소의 시퀀스내에 모든 데이터 요소에 의존한 값을 갖는다.

[0059] 또 다른 국면에서 본 본 발명의 데이터 처리방법은,

[0060] 단일 명령어 다중 데이터 오퍼랜드를 단일 명령어 다중 데이터 레지스터 파일에 기억시키는 단계;

[0061] 단일 명령어 다중 데이터 프로그램 명령어의 제어하에서 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터내에 별도의 레인내에 기억된 별도의 데이터 요소에 관해서 독립적으로 처리연산을 행하는 단계; 및

[0062] 추가의 프로그램 명령어의 제어하에서, 내부에 기억된 출력 오퍼랜드를 발생시키기 위해 상기 단일 명령어 다중 데이터 레지스터 파일의 입력 오퍼랜드용 레지스터와 상기 단일 명령어 다중 데이터 레지스터 파일의 출력 오퍼랜드용 레지스터내에 유지된 데이터 요소의 시퀀스를 포함하는 벡터 데이터 값에 관한 추가의 처리연산을 행하는 단계로서, 상기 출력 오퍼랜드의 제1 부분이 상기 데이터 요소의 시퀀스내에 모든 데이터 요소에 의존한 값을 갖는, 단계를 포함한다.

[0063] 본 발명의 또 다른 국면은, 상기 상술한 것과 같은 프로그램 명령어를 마치 상기 상술한 데이터 처리장치상에서 실행중이었던 것처럼 실행 가능하게 하는 범용 컴퓨터의 실행 환경을 제공하는 가상 머신 구현을 제공하는 것이다. 여기서는, 본 기술들의 이러한 가상 머신 구현을 포함한다.

도면의 간단한 설명

[0064] 아래의 첨부도면을 참조하여 예시로만 본 발명의 실시예들을 설명하겠다:

도 1은 단일 명령어 다중 데이터 레지스터 파일과, 암호처리 명령어들의 실행을 위한 지원을 포함하는 단일 명령어 다중 데이터 처리회로를, 구비한 데이터 처리장치를 개략적으로 나타내고;

도 2는 일 예시 형태의 해시 알고리즘내의 데이터 흐름을 개략적으로 나타내고;

도 3은 상기 추가의 처리 명령어가 단일 명령어 다중 데이터 처리회로에 관한 통상의 레인 기반 처리를 추종하지 않는 모습을 개략적으로 나타낸다.

도 1은 조작되는 데이터와 실행되는 프로그램 명령어를 기억하는 메모리(6)에 연결된 중앙처리장치(4) 형태의 데이터 처리장치(2)를 개략적으로 나타낸 것이다. 상기 중앙처리장치(4)는, 범용 레지스터 파일(8), 범용 처리회로(10), 단일 명령어 다중 데이터 레지스터 파일(12) 및 단일 명령어 다중 데이터 처리회로(14)로 이루어진다. 상기 범용 레지스터 파일(8)은, 영국 캠브리지의 ARM사에서 생산한 프로세서의 상기 범용 레지스터 파일에 의해 지원된 형태의 레지스터 등의 저비트 폭 범용 레지스터(32 또는 64비트 등)를 포함하는 것이 전형적이다. 단일 명령어 다중 데이터 레지스터 파일(12)은 매우 큰 레지스터를 구비하는 것이 전형적이고, 단일 명령어 다중 데이터 레지스터 파일(12)내의 상기 데이터 스토리지는 상이한 방식으로 분할되어 사용된 상기 레지스터 사이즈 지정자에 따라 상이한 레지스터를 형성하기도 한다. 단일 명령어 다중 데이터 레지스터 파일(12)의 형태는, 영국 캠브리지의 ARM사에서 생산한 프로세서의 일부의 구현으로 지원된 신규 레지스터 파일의 형태이어도 된다.

범용 레지스터 재명명 및 스코어 보딩 회로 16은 범용 레지스터 파일(10)과 관련되어 있고, 단일 명령어 다중 데이터 레지스터 재명명 및 스코어 보딩회로 18은 단일 명령어 다중 데이터 레지스터 파일(12)과 관련되어 있다. 레지스터 재명명 및 스코어 보딩 그 자체는, 본 기술분야에서의 작업자에게 잘 알려져 있어 여기서는 추가로 설명하지 않는 공지된 기술이다. 상기 레지스터 재명명 및 스코어 보딩은, 통상의 단일 명령어 다중 데이터 처리 명령어에 대해 제공된 것과 같은 방식으로 아래에 설명한 암호 처리 명령어의 지원에 사용된 상기 레지스터에 적용되어도 된다. 따라서, 이미 레지스터 재명명, 명령어 스케쥴링, 명령어 발행, 명령어 폐기 및 명령어 인터럽트를 지원하도록 제공된 상기 메카니즘은, 암호 지원 프로그램 명령어에 의해 재사용되어도 됨에 따라서, 이들 암호 지원 명령어의 연산은 중앙처리장치(4)의 전체 연산과 보다 좋게 통합되어도 된다.

프로그램 명령어 I는, 메모리(6)로부터 수신되어 명령어 파이프라인(20)을 통과한다. 명령어 디코더(22)는, 그 프로그램 명령어를 복호화하여, 상기 중앙처리장치(4)내의 다른 요소들뿐만 아니라 상기 레지스터 파일(8,12)과 상기 처리회로(10,14)의 동작도 제어하는 제어신호를 생성한다. 인터럽트 회로(24)는, 외부적으로 생성된 인터럽트 신호들 int에 응답하여, 현재 상기 중앙처리장치(4)에서 행하고 있는 처리를 중단하고 본 기술분야의 것들에 잘 알려져 있듯이 인터럽트 핸들링 코드의 실행을 시작한다. 중앙처리장치(4)는 다수의 추가의 회로요소를 구비하는 것이 일반적이고, 이들은 명료함을 기하기 위해서 도 2로부터 생략되어 있다는 것을 알 것이다.

도 2는 일 형태의 해시 생성 알고리즘내에서의 데이터 흐름을 개략적으로 나타낸 것이다. 해시될 파일(26)은, 64바이트 블록(28)으로 분할되고 일 입력으로서 해시 알고리즘(30)에 공급된 4개의 32비트 워드의 입력 벡터로 한층 더 분할된다. 또한, 해시 종자값은, 해시 연산의 시작부분에서 제공된다. 그 해시 연산은, 각각 상기 해시 갱신 루프(34)와 상기 스케쥴 갱신 루프(36)를 책임지는 2개의 메인 프로그램 루프를 이용한다. 이들의 메인 루프는, 상기 단일 명령어 다중 데이터 처리회로(14)내에서 양쪽의 루프를 지원하는 전용 프로그램 명령어의 제공에 의해 균형이 맞추어진다. 중간 해시값(38)은, 해시 갱신 루프(34)에 의해 생성되어 상기 해시 알고리즘으로서 피드백되어 입력 데이터의 블록(28)을 계속 처리한다. 그 블록(28)이 처리되었을 때(예를 들면, SHA-1일 경우에 80번의 해시 갱신이 반복됨), 출력 해시값(40)은 현재의 중간 해시값(38)에 가산하여서 갱신된다. 이 처리는, 상기 파일(26) 모두가 완료될 때까지 반복된다. 이것은, 전체적으로 상기 파일(26)에 대한 결과 해시값을 생성한다.

상기 해시 갱신 루프(34)는, 여러 번 실행되고, 후술하는 것처럼 각각 명령어들 자신의 명령어내 반복을 하게 하는 그 명령어들을 실행한다. 상기 스케쥴 갱신 루프(36)는, 상기 해시 갱신 루프와 균형을 맞추도록 행해진다. 상기 스케쥴 갱신 루프는 후술하는 것처럼 성능을 향상시키기 위해 벡터화되어도 된다.

도 3은 본 기술에 따라 추가의 처리 명령어가 복수의 데이터 요소를 포함한 벡터 데이터 값(42)을 수신하는 모습을 개략적으로 나타낸 것이다. 그리고, 암호 지원 명령어는, 이 벡터 데이터 값(42)에 관한 처리 연산을 행하여, 상기 벡터 데이터 값(42)의 제1 데이터 요소와 상기 벡터 데이터 값내의 2개 이상의 추가의 데이터 요소 양쪽에 의존하는 제1 부분(44)을 갖는 출력 오퍼랜드를 생성한다. 이러한 작용은, 상기 처리연산이 레인 기반 연산이고, 상이한 레인내에서 데이터 값간에 어떠한 상호작용일 경우도 제한이 있는 전형적인 단일 명령어 다중 데이터 프로그램 명령어들과 대조한다.

본 기술의 일 구현은, 2개의 알고리즘 즉, SHA-1 및 SHA-256을 대상으로 하는 일 세트의 명령어이다. 또한, 그 명령어는 SHA-256과 같은 연산을 필요로 하는 SHA-224 알고리즘에 도움을 준다. SHA 알고리즘은, 미국 국립 표준 기술 연구소(NIST)에서 규정한 안전한 해시 알고리즘의 패밀리다. 이들 알고리즘의 사양은, 개방적으로 이용 가능하다. 그 알고리즘은, 디지털 시스템내에서 데이터를 인증하는데 사용되는 것이 일반적이다.

우리는, 상기 SHA 알고리즘의 하이레벨 연산을 기술하고 SHA-1 및 SHA-256 알고리즘에 대한 의사코드를 포함하여서 개시한다.

SHA 알고리즘의 하이레벨 연산(공지됨; FIPS 180-4)

상기 알고리즘 각각은 64바이트의 데이터를 처리하고, 해시 다이제스트(digest)를 발생시킨다; SHA-1의 경우에, 이것은 길이가 160비트이고, SHA-256의 경우에, 이것은 길이가 256비트다. 64바이트보다 큰 길이의 데이터 스트림은, 64바이트 블록으로 분할된다. 스트림이나 블록이 길이가 64바이트미만인 경우, 그 블록은, FIPS(Federal Information Processing Standard) 180-4에 규정된 것처럼, 64바이트로 채워넣어진다. 달리 언급하지 않으면, 그 알고리즘에 대한 이하의 설명은, 워드를 32비트의 비부호 정수값으로 가정한다. 워드가, 빅 엔디언 형태로 64바이트의 블록으로부터 4개의 인접한 바이트로 이루어진다고 가정한다.

양쪽의 알고리즘은 워킹(working) 해시 다이제스트를 초기화함으로써 시작한다. 그 데이터의 블록이 주어진 데이터 스트림에서 첫 번째일 경우, 그 해시 다이제스트는 고정된 종자값으로 초기화된다. 그 블록이 데이터 스트림의 연속일 경우, 상기 해시 다이제스트는 이전의 블록으로부터 산출된 상기 해시 다이제스트에 초기화된다. 상기 종자값은 FIPS 180-4에 규정되어 있다.

상기 알고리즘은 스케쥴 갱신 연산을 이용하여 상기 블록을 확장하다. 이것은, SHA-1의 경우 초기의 16개의 워드로부터 80개의 워드의 데이터로 상기 블록을 확장하고, SHA-256의 경우에는 64개의 워드의 데이터로 상기 블록을 확장한다. 상기 스케쥴 갱신 연산은, 고정된 xor, 시프트 및 회전을 사용하여, 상기 스케쥴로부터 4개의 워드를 조합하여, 상기 확장된 스케쥴에서 다음의 워드를 생성한다. 초기의 16개의 워드는, 상기 확장 스케쥴에서 그대로 남아 있다.

그리고, 상기 확장 스케쥴에서 각 워드는, 키값이 가산되어 있다. SHA-1에는, 상기 확장 블록으로부터 일 세트의 20개의 워드에 각각 적용된 키 상수 4개가 있다. SHA-256에는, 상기 확장 블록의 워드마다 키 상수 64개가 있다. 그 키 상수는, FIPS 180-4에 규정되어 있다. 그 블록을 확장하고 키 상수를 가산한 후에, 각 워드는, 일련의 고정된 xor, 시프트 및 회전에 의해 그 워드를 상기 해시 다이제스트에 포함하는 해시 갱신함수를 사용하여 처리된다.

끝으로, 상기 확장 블록으로부터의 각 워드가 상기 해시 갱신 함수를 이용하여 처리된 후에는, 상기 해시 다이제스트가 이전의 해시 다이제스트 값에 가산된다.

FIPS 180-4에 규정된 바와 같이, 상기 스케쥴은, 일 세트의 80/64개의 워드(SHA-1/SHA-256)로서 또는 원형 큐의 16개의 워드로서 구현될 수 있다.

SHA-1 및 SHA-256에 대한 의사코드 알고리즘을 완료하기 위해서, 원형 큐를 가정한 것이 아래에 표현되어 있다.

SHA-1 알고리즘 의사코드

```
uint32 w[0:15] = 16 4-bytes (big-endian) input[];
```

```
uint32 wk[0:15] = w[0:15] + k[0:15];
uint32 a:e = io->hashes[0:4];
for round=0:63 {
hash_update(round,wk[round]);
w[round] = schedule_update(round,w);
wk[round ] = w[round] + k[round];
}
for round=64:79
hash_update(round,w[round]);
io->hashes[0:4] += a:e;
```

The SHA-1 hash update code being as follows:

```
hash_update(int round, uint32 wk) {
e += FN( round,b,c,d) + ROL(a,5) + wk;
b = ROL(b,30);
rotate (a,b,c,d,e) to (e,a,b,c,d)
}
where:
if round < 20, FN = choose(b,c,d);
else if round < 40, FN = parity(b,c,d);
else if round < 60, FN = majority(b,c,d);
else FN = Parity(b,c,d);
choose(b,c,d) = (((c ^ d) & b) ^ d)
parity(b,c,d) = (b ^ c ^ d)
majority(b,c,d) = (b & c) | ((b | c) & d)
```

The SHA-1 schedule update code being as follows:

```
uint32 schedule_update(int round, uint32 *w) {
return ROR(w[round-3] ^ w[round-8] ^ w[round-14] ^ w[round-16], 31);
}
```

SHA-256 알고리즘 의사코드

```
uint32 a:h = io->hashes[0:7];
uint32 w[0:15] = 16 4-bytes (big-endian) input[0:63];
uint32 wk[0:15] = w[0:15] + k[0:15];
for round=0:47 {
hash_update(wk[round]);
w[round] = schedule_update(round,w);
 wk[round] = w[round] + k[round];
 }
 for round=48:63
 hash_update(wk[round]);
 io->hashes[0:7] += a:h;
```

SHA-256 해시 갱신 코드가 다음과 같다:

```
 hash_update(uint32 wk) {
 t = h + Sigma1(e) + Choose(e,f,g) + wk;
 d += t;
 h = t + Sigma0(a) + Majority(a,b,c);
 rotate (a,b,c,d,e,f,g,h) to (h,a,b,c,d,e,f,g);
 }
```

여기서,

```
Sigma0(x) = ror(x,2) ^ ror(x,13) ^ ror(x,22);
Sigma1(x) = ror(x,6) ^ ror(x,11) ^ ror(x,25);
Choose(b,c,d) = (((c ^ d) & b) ^ d)
Majority(b,c,d) = ((b & c) | ((b | c) & d)
```

마찬가지로, SHA-256 스케쥴 갱신 의사 코드는 다음과 같다:

```
uint32 schedule_update(int round, uint32 *w) {
return w[round] + sigma1(w[round-2]) + w[round-7] + sigma0(w[round-15]);
}
```

여기서,

```
sigma0(x) = ror(x,7) ^ ror(x,18) ^ shr(x,3);
sigma1(x) = ror(x,17) ^ ror(x,19) ^ shr(x,10);
```

SHA 알고리즘 워킹 상태(FIPS 180-4 사양으로부터 얻어질 수 있음)

상기 SHA 알고리즘을 가속하는데 취한 해결방법을 제약하는 일 국면은, (이전에 설명한 바와 같이) 일 블록의 데이터를 처리하는데 필요한 많은 워킹 상태다. 단일 명령어 다중 데이터 레지스터 파일을 이 상태를 유지 및 조작하는 능력은 상기 제약을 해결한다.

다음의 표는, SHA-1 및 SHA-256에 대한 상태 요구사항의 개요를 나타낸다.

SHA-1 상태

초기/이전의 해시 다이제스트	5×32비트 워드
워킹 해시 다이제스트	5×32비트 워드
스케쥴	16×32비트 워드
키 상수	4×32비트 워드

SHA-256 상태

초기/이전의 해시 다이제스트	8×32비트 워드
워킹 해시 다이제스트	8×32비트 워드
스케쥴	16×32비트 워드
키 상수	64×32비트 워드

SHA-1 또는 SHA-256의 알고리즘을 사용하여, 일 블록의 데이터를 처리 가능한 (예를 들면, 코프로세서로서) 전용 SHA 유닛을 구축하려면, 확고한 목적의 상태에서 투자를 필요로 한다. 이 상태는 RISC 마이크로프로세서상에서 다른 연산에서 쉽게 사용될 수 있었다.

SHA 알고리즘들을 3개 한 벌의 형태의 RISC 명령어에 넣기

확고한 목적의 상태를 피하기 위해서, 우리는, 상기 3개 한 벌의 명령어 형태를 준수하며 단일 명령어 다중 데이터 처리회로와 레지스터 파일을 사용하여 상기 알고리즘들을 처리할 수 있는 방식으로 그 알고리즘들을 분할하였다.

상기 RISC 3개 한 벌의 형태의 전형적인 제약은, 3개의 레지스터 중 하나의 레지스터만을 수신지로서 정의하는 것이다. 그렇지만, 상기 수신지는 소스로서 사용될 수 있다.

우리는 SIMD 레지스터를 사용함으로써 명령어마다 범용 레지스터를 사용하여 처리 가능한 것보다 많은 데이터를 처리할 수 있다.

상기 3개 한 벌의 명령어 형태를 준수함으로써, 그 명령어는 최신의 마이크로프로세서에 공통된, 리네임(rename), 스케쥴링, 발행, 결과 및 폐기 로직을 이용할 수 있다.

모든 상태와 종속성이 그 명령어에 의해 규정되어 있으므로, 비순차적 실행, 인터럽션 및 투기적 실행(speculation)을 처리하는 파이프라인 메카니즘이 그래도 유효하고; 상기 제안된 명령어들의 정확한 실행을 유

지하는데 제어 로직을 추가할 필요가 없다.

SHA-1 해시 갱신 명령어

이전에 설명한 것처럼, SHA-1 해시 갱신함수는, 32비트 워드를 160비트 해시 다이제스트에 넣는다. 이 함수는 고정된 시프트, 고정된 xor/and/or 및 고정된 회전으로 이루어진다.

```
hash_update(int round, uint32 wk) {
e += FN( round,b,c,d) + ROL(a,5) + wk;
b = ROL(b,30);
rotate (a,b,c,d,e) to (e,a,b,c,d)
}
```

여기서,

```
if round < 20, FN = choose(b,c,d);
else if round < 40, FN = parity(b,c,d);
else if round < 60, FN = majority(b,c,d);
else FN = Parity(b,c,d);
choose(b,c,d) = (((c ^ d) & b) ^ d)
parity(b,c,d) = (b ^ c ^ d)
majority(b,c,d) = (b & c) | ((b | c) & d)
```

SHA-1 해시 다이제스트는 160비트이기 때문에, 전체 다이제스트 플러스 32비트 워드에 관해 행하는 연산은 32비트 범용 3개 한 벌의 RISC형태로 가능하지 않고 64비트 범용 3개 한 벌의 RISC형태로 실현하는데 상당한 노력을 요할 것이고; 32비트 데이터 값을 제3의 64비트 오퍼랜드의 하이 32비트에 삽입하는데 보다 많은 정리 작업(housekeeping)이 필요할 것이다.

이 때문에, 본 예시 기술은 SHA-1 해시 함수를 개선된 일 세트의 4개의 SIMD 명령어, 즉 SHA1C, SHA1P, SHA1M 및 SHA1H에 매핑한다.

```
SHA1C Qd, Sn, Vm.4S [OP = C, OP_FUNC = choose]
SHA1P Qd, Sn, Vm.4S [OP = P, OP_FUNC = parity]
SHA1M Qd, Sn, Vm.4S [OP = M, OP_FUNC = majority]
SHA1H Sd, Sn
```

명령어 SHA1C, SHA1P 및 SHA1M은 3개의 오퍼랜드를 취한다. Qd는 상기 다이제스트 해시의 첫번째 4개의 32비트 워드를 보유하고, 이때 Sn은 다섯 번째를 보유한다. 세 번째 오퍼랜드 Vm은, 초기의 실시예에서는 4개의 32비트 워드를 보유하는 벡터다. 이에 따라 그 해시 갱신 함수를 상기 명령어에 의해 4번 반복 처리할 수 있다. 의사코드는, 이들 명령어의 연산이 아래에서 표현되도록 정의되어 있다. 의사코드의 관점에서 어떤 명령어의 연산을 정의하는 것은 본 기술분야에 잘 알려져 있고, 상기 의사코드에 의해 정의된 명령어를 행하는(실행하는) 회로의 실현은, 일단 상기 의사코드가 정의되어 있었다면 일상적인 것이라는 것을 알 것이다.

```
SHA1<OP> Qd, Sn, Vm.4S
  X = Qd;
  Y = Sn;
  for (i = 0 to 3)
  {
    Index          = (i * 32);
    t1<31:0>       = OP_FUNC(X<63:32>, X<95:64>, X<127:96>);
    Y              = Y + ROL(X<31:0>, 5) + t1<31:0> + Vm<(index + 31):index>;
    X<63:32>       = ROL(X<63:32>, 30);
    // Rotate
    t2<31:0>       = Y;
    Y              = X<127:96>;
    X<127:0>       = {X<95:0>:t2<31:0>};
  }
Qd = X;
```

따라서, 일 실시예의 예시에 따라, 상기 단일 명령어 다중 데이터 처리회로는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0]를 갖는 추가의 프로그램 명령어에 의해 제어되도록 구성되고, 상기 벡터 데이터 값이 N이 양의 정수인 $Vm[32*2^{N}-1: 0]$를 포함하고, 상기 추가의 처리 연산이 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드Qd_{output}[127:0]를 발생시킨다:

```
X[127:0] = Qd[127:0];
Y[31:0] = Sn[31:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                t1[31:0] = OP FUNC (X[63:32], X[95:64], X[127:96]);
                Y[31:0] = Y[31:0] + ROL(X[31:1], 5) + T1[31:0] + Vm[Index+31:Index];
                X[63:32] = ROL(X[63:32], 30);
                T2[31:0] = Y[31:0];
                Y[31:0] = X[127:96];
                X[127:0] = {X[95:0]:T2[31:0]}
        }
Qd_output[127:0] = X[127:0];
```

여기서, OP FUNC(B,C,D)는

(((C XOR D) AND B)XOR D);

(B XOR C XOR D); 및

(B AND C) OR ((B OR C) AND D)중 하나이고,

ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

이들 명령어의 다른 실현은, choose(), parity() 및 majority()함수 중에서 선택하기 위한 선택을 포함

한다:

```
SHA1HASH    Qd, Sn, Vm.4S, #OP // #OP where #1 selects C, #2 selects P, #3 selects M.
```

상기 RISC명령어 형태의 제약은, 상기 해시 다이제스트의 첫 번째 4개의 워드만을, SHA1C, SHA1P 및 SHA1M 명령어에 의해 128비트 레지스터 Qd에 돌려줄 수 있는 것이다. 이 때문에, 명령어 SHA1H는, 상기 해시 다이제스트의 다섯 번째 워드를 돌리주도록 제안된다.

초기의 실현에서는 SHA1H를 다음과 같이 구현한다:

```
SHA1H Sd, Sn
  Sd      = ROR(Sn, 2);
```

4번의 반복 후의 다섯 번째 해시 다이제스트 값이 Qd[0]의 초기값의 회전이다는 것을 준수한다.

SHA1 해시 갱신 명령어 변형

SHA1C, SHA1P 및 SHA1M 명령어의 변형은, 본 기술의 다른 변형에 의해 확장되어 Vm.8S 오퍼랜드 또는 Vm.16S 오퍼랜드를 허용할 수 있었다. 이들의 변형은, 본 기술내에 포함된다. 이에 따라 단일 명령어내에서 상기 해시 갱신 함수를 8번 및 16번 반복 처리할 수 있다. 그렇긴 하지만, 상기 Vm.4S 변형은, 매 20번 반복 후에 상기 해시 갱신 함수를 변경시킬 필요가 있으므로 그래도 필요할 것이다.

일례로서 상기 SHA1<OP>Vm.8S 변형:

```
SHA1<OP> Qd, Sn, Vm.8S
  X = Qd;
  Y = Sn;
  for (i = 0 to 7)
  {
    Index          = (i * 32);
    t1<31:0>       = OP_FUNC(X<63:32>, X<95:64>, X<127:96>);
    Y              = Y + ROL(X<31:0>, 5) + t1<31:0> + Vm<(index + 31):index>;
    X<63:32>       = ROL(X<63:32>, 30);
    // Rotate
    t2<31:0>       = Y;
    Y              = X<127:96>;
    X<127:0>       = {X<95:0>:t2<31:0>};
  }
Qd = X;
```

8번 및 16번의 반복에 걸쳐 연산하는 변형(Vm.8S 및 Vm.16S)은, 추가로 SHA1C2, SHA1P2 및 SHA1M2 명령어를 필요로 한다. 이들 명령어는, 8번 반복이나 16번 반복 후에 상기 해시 다이제스트에서 다섯 번째 워드에 대한 적절한 값을 발생시킨다. 이들의 새로운 명령어는, SHA1C, SHA1P 및 SHA1M 명령어와 마찬가지의 방식으로 구현되지만, 예를 들면, 아래와 같은 Qd 레지스터에서 다섯 번째 해시 다이제스트 워드를 돌려준다:

```
SHA1<OP> 2 Qd, Sn, Vm.8S
 X = Qd;
 Y = Sn;
 for (i = 0 to 3)
 {
  Index        = (i * 32);
  t1<31:0>     = OP_FUNC(X<63:32>, X<95:64>, X<127:96>);
  Y            = Y + ROL(X<31:0>, 5) + t1<31:0> + Vm<(index + 31):index>;
  X<63:32>     = ROL(X<63:32>, 30);

  // Rotate
  t2<31:0>     = Y;
  Y            = X<127:96>;
  X<127:0>     = {X<95:0>:t2<31:0>};
 }
Qd = {0:Y<31:0>};
```

따라서, 일 실시예의 예시에 따라, 상기 단일 명령어 다중 데이터 처리회로는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Sn[31:0]를 갖는 추가의 프로그램 명령어에 의해 제어되도록 구성되고, 상기 벡터 데이터 값이 N이 양의 정수인 $Vm[32*2^{N}-1: 0]$를 포함하고, 상기 추가의 처리 연산이 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드Qd_{output}[127:0]를 발생시킨다.

```
X[127:0] = Qd[127:0];
Y[31:0] = Sn[31:0];
for (I = 0 to (2^N-1));
        {
                Index    = (I * 32);
                T1[31:0]  = OP_FUNC(X[63:32], X[95:64], X[127:96]);
                Y        = Y + ROL(X[31:0], 5) + T1[31:0] + Vm[(Index + 31):Index];
                X[63:32]  = ROL(X[63:32], 30);
                T2[31:0]  = Y;
                Y        = X[127:96];
                X[127:0]  = {X[95:0]:T2[31:0]};
        }
Qd_output[127:0] = {0:Y[31:0]};
```

여기서, OP FUNC(B,C,D)는

(((C XOR D) AND B)XOR D);

(B XOR C XOR D); 및

(B AND C) OR ((B OR C) AND D)중 하나이고,

ROL(P,Q)는 P값의 Q비트 위치만큼의 좌회전이다.

상기 명령어의 다른 변형은, 전체 해시 다이제스트를 보다 넓은 SIMD 데이터패스가 하기를 이용 가능하였다면 8워드(8x32비트)로 되돌리도록 실현될 수 있었다:

SHA1<OP>　　Qd,Vn.4S

SHA1<OP>　　Qd,Vn.8S.

이들의 명령어는, 상기 해시 함수의 4번 및 8번 반복을 처리한다.

SHA1 해시 갱신 마이크로 구조 선택사항

각 종 선택사항은 이들 명령어의 마이크로구조 실현에 대해 존재한다:

이들 명령어의 고성능 실현을 선택하여, 반복 로직 중 일부를 확장하고 보다 많은 병행 실행을 행하여도 된다.

상기 마이크로 구조를 선택하여 일시적 파이프라인 상태와 그에 따른 소비전력을 감소시키기 위해서 멀티 사이클 단(stages)을 이용할 수 있었다.

중간 계산은 자리올림 보존 형태로 행해질 수 있다.

명백한 SHA1<OP>2 명령어가 필요한 경우의 보다 넓은 변형에서는, SHA1<OP>2 연산이 대응한 SHA1<OP>함수에 후속할 때 검출하는 것이 가능하기도 하다. 그들의 경우에는, 두 번째 계산을 방지하고 상기 데이터패스로부터의 결과를 간단히 보내는 것이 가능해야 한다. 이에 따라 파이프라인에서 임시의 상태를 일부 필요로 한다.

SHA1 스케쥴 갱신 명령어

SHA1 알고리즘으로 속도를 향상시키려면 상기 해시 갱신함수와 스케쥴 갱신 함수간에 균형이 필요하다.

이전에 설명한 것처럼, SHA1 스케쥴 갱신 함수는, 상기 데이터 스케쥴로부터의 4개의 32비트 워드를 상기 스케쥴을 확장하는 단일 결과 워드로 조합하거나, 원형 큐일 경우에는, 상기 스케쥴에 하나의 워드를 겹쳐쓴다.

그 스케쥴 갱신 연산은, xor들과 고정된 회전으로 이루어진다.

```
uint32 schedule_update(int round, uint32 *w) {
return ROR(w[round-3] ^ w[round+8] ^ w[round-14] ^ w[round-16], 31);
}
```

or in the circular queue form:

```
void schedule_update(int round, uint32 w[0..15]) {
w[round] = ROR(w[round+13mod16] ^ w[round+8mod16] ^ w[round+2mod16] ^ w[round], 31);
}
```

이 연산은 4개의 입력값을 필요로 하고, 이 입력값 중 하나는 파괴적이다. 이것은, 범용 32- 3개 한 벌의 RISC 형태에 맞지 않는다.

상기 스케쥴 갱신 명령어는, ARM사의 개선된 SIMD 아키텍처에 의해 제공되어도 된다.

메모리 로드(load)와 스토어(store)를 피하기 위해서, 우리는, FIPS 180-4에 기재된, 상기 스케쥴 갱신의 원형 큐 형태를 효율적으로 실행하는 명령어를 구현하도록 선택했다.

완전을 기하기 위해서, 우리는 스케쥴 갱신의 벡터화 방법을 포함한다.

SHA-1 스케쥴 갱신 벡터화 및 치환

상기 준수의 결과로서, w[round], w[round+1_{mod16}] 및 w[round+2_{mod16}]는 동시에 처리될 수 있다. 4-웨이

(way) 벡터화에 대한 직접 경로를 방지하는 w[round+3$_{mod16}$]의 계산에서는 w[round]에 의존한다.

```
w[round  ] = ROR(w[round+13] ^ w[round+8 ] ^ w[round+2] ^ w[round  ]), 31);
w[round+1] = ROR(w[round+14] ^ w[round+9 ] ^ w[round+3] ^ w[round+1]), 31);
w[round+2] = ROR(w[round+15] ^ w[round+10] ^ w[round+4] ^ w[round+2]), 31);
w[round+3] = ROR(w[round  ] ^ w[round+11] ^ w[round+5] ^ w[round+3]), 31);
```

이러한 제약은, w[round+3$_{mod16}$]의 계산에서 w[round]의 값에 대해 0으로 치환하여, 그 결과를 추가의 xor과 회전의 단계로 결정함으로써 해결될 수 있다; 이것은 아래에 설명되어 있다.

```
w[round  ] = ROR(w[round+13] ^ w[round+8 ] ^ w[round+2] ^ w[round  ]), 31);
w[round+1] = ROR(w[round+14] ^ w[round+9 ] ^ w[round+3] ^ w[round+1]), 31);
w[round+2] = ROR(w[round+15] ^ w[round+10] ^ w[round+4] ^ w[round+2]), 31);
w[round+3] = ROR(         0 ^ w[round+11] ^ w[round+5] ^ w[round+3]), 31);
w[round+3] = w[round+3] ^ ROR(w[round], 31);
```

상기 블록의 코드를 리팩토링 하여(re-factor) 데이터패스의 사이즈가 4×32비트인 SIMD 아키텍처상에서 4레인 벡터 연산을 이용할 수 있다.

SHA-1 스케쥴 갱신 및 해시 갱신 균형 맞추기

상기 스케쥴 갱신 연산과 상기 해시 갱신 연산의 균형을 맞추기 위해서, 상기 스케쥴 갱신은, 이전에 설명한 것처럼, 즉 4-웨이 벡터화를 사용하여 처리된다. 이에 따라 단일 스케쥴 갱신을 할 수 있어 이후의 해시 함수 명령어에 대해 충분한 데이터, 4×32비트 워드를 발생시킨다.

상기 벡터화 기술은, 합리적인 SIMD 구현에 있어서, 상기 제안된 SHA-1 해시 함수를 실행하는데 걸리는 실행 사이클보다 상기 스케쥴 데이터를 계산하는데 많은 실행 사이클이 걸릴 것이다.

이것에 대해서는 이유가 여러 가지다:

요소 {round+2, round+3, round+4, round+5}를 포함하는 벡터는 2개의 벡터 레지스터에 걸쳐 있을 가능성이 있다.

요소 {round+13, round+14, round+15, 0}를 포함하는 벡터는 하나의 벡터 레지스터와 제로 벡터로부터 추출될 필요가 있다.

SIMD 벡터 회전은, SIMD 명령어 세트, 예를 들면 ARM 개선 SIMD명령어에서는 일반적으로 발견되지 않고 있다. 그래서, 벡터 회전은, 2개의 벡터 시프트와 하나의 or 명령어를 필요로 한다.

암달의 법칙(Amdahl's law)으로 인해서, SHA-1 알고리즘의 양쪽 부분은 균형을 맞추어야 하고, 그렇지 않으면 부품의 속도가 느릴수록 달성 가능한 많은 속도향상이 제한될 것이다.

이러한 소견에 따라 SHA-1 스케쥴 갱신 함수를 가속하기 위한 다음의 SIMD 명령어가 행해지게 된다.

```
SHA1SU0 Vd.4S, Vn.4S, Vm.4S
  T<127:0>      = Vn<63:0>:Vd<127:64>
  Vd            = T XOR Vd XOR Vm
SHA1SU1 Vd.4S, Vn.4S
```

T<127:0> = Vd XOR {32{0}:Vn<127:32>};

Vd<95:0> = T<94:64>:T<95>:T<62:32>:T<63>:T<30:0>:T<31>;

Vd<127:96> = (T<126:96>:T<127>) XOR (T<29:0>:T<31:30>);

이 명령어는, 상기 원형 큐에는 4개의 4×32비트 벡터 레지스터가 있다는 것을 가정한다.

요소들의 재배치는 상기 명령어 내부에서 제어된다. 이에 따라 효율적으로 상기 요소들의 재배치를 자유롭게 하고, 그것들은 마이크로 구조에서 배선들일뿐이다.

고정된 회전도 배선들일뿐이다.

상기 명령어는, 균형이 맞추어지고, 대부분의 마이크로 구조에서 사이클 대기시간이 매우 짧아도 되고; 그들은 직렬 배선으로 2개의 xor를 포함한다.

따라서, 실시예의 예시에 따라, 상기 단일 명령어 다중 데이터 처리회로는, 제1 입력 오퍼랜드 Sp[127:0]와 제2 입력 오퍼랜드Sq[127:0]을 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sr[127:0]를 생성하는, 제1 스케쥴 명령어에 의해 제어되도록 구성된다:

T[127:0]={Sp[63:0]:Sq[127:64]} 및

Sr[127:0]=T[127:0] XOR Sr[127:0] XOR Sq[127:0].

SHA-2 알고리즘을 대상으로 하는 명령어

상기 SHA-1 알고리즘을 위해 제안된 명령어의 설명에서 개요를 나타낸 많은 특징은, SHA-2 알고리즘에도 마찬가지로 적용한다. 본 절에서는, SHA-2 알고리즘을 위해 제안된 명령어에서의 차이점을 설명한다.

SHA-2 해시 갱신 명령어

SHA-1에 대해 개요를 나타낸 그 이유 때문에, 2개의 해시 갱신 명령어에서는 SHA-2 해시 갱신 함수를 대상으로 한다.

SHA-2 알고리즘에 대해서 워킹 해시 다이제스트는, 256비트 또는 512비트다. 이하에서는, 본 발명의 초기 실현시에 포함되는 것처럼, 256비트의 워킹 해시를 갖는 SHA-256 알고리즘과 SHA-224 알고리즘에 초점을 둔다. 후자의 절에서는, 본 기술을 SHA-512, SHA-384, SHA-512/256 및 SHA-512/224에 적용하는 방법을 설명한다.

SHA-256 해시 갱신 명령어

SHA-256(및 SHA-224)의 워킹 해시 다이제스트의 길이는 256비트이다. 레지스터 폭이 128비트인 SIMD아키텍처에서는, 상기 해시 다이제스트에 관해 어떠한 조작에 의한 결과도 2개의 명령어, 즉, 첫 번째 4×32비트 워드를 돌려주는 명령어와 그 나머지 4×32비트 워드를 돌려주는 명령어를, 필요로 한다.

SHA-1과 달리 상기 SHA-2 해시 갱신 함수는, 고정되어 있고, 주어진 수의 반복 후에 변경하지 않으므로, 2개의 명령어만을 필요로 한다.

SHA256H Qd, Qn, Vm.4S

X = Qd;

Y = Qn;

```
for (i = 0 to 3)
{
        index           = (i * 32);
        tCh<31:0>       = Choose(Y<31:0>, Y<63:32>, Y<95:64>);
        tMaj<31:0>      = Majority(X<31:0>, X<63:32>, X<95:64>);
    t1<31:0>    = Y<127:96> + Sigma1(Y<31:0>)
                + tCh<31:0>  + Vm<(index + 31):index>;
        X<127:96>       = t1<31:0> + X<127:96>;
        Y<127:96>       = t1<31:0> + Sigma0(X<31:0>) + tMaj<31:0>;
        t2<31:0>        = Y<127:96>;
        Y<127:0>        = Y<95:0>:X<127:96>;
        X<127:0>        = X<95:0>:t2<31:0>;
}
Qd = X;
```

따라서, 일 실시예의 예시에 따라, 상기 단일 명령어 다중 데이터 처리회로는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0]를 갖는 추가의 프로그램 명령어에 의해 제어되도록 구성되고, 상기 벡터 데이터 값이 N이 양의 정수인 $Vm[32*2^{N}-1:0]$를 포함하고, 상기 추가의 처리 연산이 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드Qd_{output}[127:0]를 발생시킨다.

```
X[127:0] = Qd[127:0];
Y[127:0] = Qn[127:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);
                TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);
                T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];
                X[127:96] = T1[31:0] + X[127:96];
                Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]
                T2[31:0] = Y[127:96];
                Y[127:0] = {Y[95:0]:X[127:96]};
                X[127:0] = {X[95:0]:T2[31:0]}
        }
Qd_output[127:0] = X[127:0];
```

여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

```
SHA256H2      Qd, Qn, Vm.4S
 X = Qn;
 Y = Qd ;
 for (i = 0 to 3)
 {
        index              = (i * 32);
        tCh<31:0>          = Choose(Y<31:0>, Y<63:32>, Y<95:64>);
        tMaj<31:0>         = Majority(X<31:0>, X<63:32>, X<95:64>);
    t1<31:0>    = Y<127:96> + Sigma1(Y<31:0>)
              + tCh<31:0>  + Vm<(index + 31):index>;
        X<127:96>          = t1<31:0> + X<127:96>;
        Y<127:96>          = t1<31:0> + Sigma0(X<31:0>) + tMaj<31:0>;
        t2<31:0>           = Y<127:96>;
        Y<127:0>           = Y<95:0>:X<127:96>;
        X<127:0>           = X<95:0>:t2<31:0>;
 }
 Qd = Y;
```

따라서, 일 실시예의 예시에 따라, 상기 단일 명령어 다중 데이터 처리회로는, 상기 단일 명령어 다중 데이터 레지스터 파일로부터 판독된 제1 입력 오퍼랜드Qd[127:0]와 제2 입력 오퍼랜드Qn[127:0]를 갖는 추가의 프로그램 명령어에 의해 제어되도록 구성되고, 상기 벡터 데이터 값이 N이 양의 정수인 $Vm[32*2^N-1:0]$를 포함하고, 상기 추가의 처리 연산이 아래의 단계로 표현된 것과 같은 값을 갖도록 상기 출력 오퍼랜드Qd_{output}[127:0]를 발생시킨다.

```
X[127:0] = Qn[127:0];
Y[127:0] = Qd[127:0];
for (I = 0 to (2^N-1));
        {
                Index = (I*32);
                TCh[31:0] = Choose(Y[31:0], Y[63:32], Y[95:64]);
                TMaj[31:0] = Majority(X[31:0], X[63:32], X[95:64]);
                T1[31:0] = Y[127:96] + Sigma1(Y[31:0]) + TCh[31:0] + Vm[Index+31:Index];

                    X[127:96] = T1[31:0] + X[127:96];
                    Y[127:96] = T1[31:0] + Sigma0(X[31:0]) + TMaj[31:0]
                    T2[31:0] = Y[127:96];
                    Y[127:0] = {Y[95:0]:X[127:96]};
                    X[127:0] = {X[95:0]:T2[31:0]}
          }
 Qd_output[127:0] = Y[127:0];
```

여기서, Choose(B,C,D)는 (((C XOR D) AND B) XOR D)이고, Majority(B,C,D)는 ((B AND C) OR ((B OR C) AND

D))이고, Sigma0(B)는 (ROR(B,2) XOR ROR(B,13) XOR ROR(B,22))이고, Sigma1(B)는 (ROR(B,6) XOR ROR(B,11) XOR ROR(B,25))이고, ROR(P,Q)는 P값의 Q비트 위치만큼의 우회전이다.

SHA256H에서는, Qd에서 상기 해시 다이제스트의 첫 번째 4×32비트 워드, Qn에서 나머지 4×32비트 워드 및 Vm.4S에서 스케쥴 데이터의 4×32비트 워드를 예상한다.

SHA256H2에서는, Qd에서 상기 해시 다이제스트의 두 번째 4×32비트 워드, Qn에서 첫번째 4×32비트 워드 및 Vm.4S에서 스케쥴 데이터의 4×32비트 워드를 예상한다.

이때, SHA256H가 상기 해시 다이제스트의 첫 번째 4×32비트 워드를 자동 파괴할 때, Qn에서 SHA256H2에 정확한 값을 건네줄 수 있도록 SHA256H를 실행하기에 앞서 사본을 만들어야 한다.

SHA-256 해시 갱신 명령어 변형

이전에 상기 SHA-1 해시 갱신 명령어에 대해 개요를 나타낸 바와 같이, 보다 넓은 벡터 SIMD의 SHA-256의 변형은 다음을 포함할 수 있었다:

SHA256(H|H2)Qd, Qn, Vm.8S

SHA256(H|H2)Qd, Qn, Vm.16S

이들 명령어는, 각각, 상기 해시 갱신함수를 8번 및 16번 반복 처리한다. 보다 넓은 SIMD 데이터패스도 다음을 허용하기도 한다:

SHA256H Qd, Vm.8S

SHA256H Qd, Vm.16S

여기서, Qd는 폭이 256비트인 레지스터이고, SHA256H2연산을 제공할 필요가 없고, 해시 다이제스트 전체는 벡터 레지스터에 들어간다.

SHA-256 스케쥴 갱신

SHA-1에 대해 이전에 개요를 나타낸 바와 같이, SHA-256 알고리즘으로 속도를 향상시키려면, 상기 해시 갱신함수와 상기 스케쥴 갱신함수간에 균형을 맞출 필요가 있다.

SHA-256 스케쥴 갱신함수는, 상기 데이터 스케쥴로부터의 4개의 32비트 워드를, 상기 스케쥴을 확장하거나, 원형 큐일 경우에 상기 스케쥴에 워드를 겹쳐쓰는 단일 결과 워드로 조합한다.

상기 스케쥴 갱신 연산은, xor, 고정된 시프트 및 고정된 회전(공지됨)으로 이루어진다.

```
uint32 schedule_update(int round, uint32 *w) {
return w[round] + sigma1(w[round-2]) + w[round-7] + sigma0(w[round-15]);
}
```

where:

sigma0(x) = ror(x,7) ^ ror(x,18) ^ shr(x,3);

sigma1(x) = ror(x,17) ^ ror(x,19) ^ shr(x,10);

This can also be expressed in a circular queue (known):

```
void schedule_update(int round, uint32 *w) {
w[round] = sigma1(w[round+14 mod16]) + w[round+9 mod16] + sigma0(w[round+1 mod16]);
}
```

SHA-256 스케쥴 갱신 벡터화 및 치환

또한, SHA-256 스케쥴 갱신 함수는, 4-웨이 SIMD에 적합한 방식으로 벡터화될 수도 있다.

```
w[round  ] = sigma1(w[round+14]) + w[round+9 ] + sigma0(w[round+1]);
w[round+1] = sigma1(w[round+15]) + w[round+10] + sigma0(w[round+2]);
w[round+2] = sigma1(w[round   ]) + w[round+11] + sigma0(w[round+3]);
w[round+3] = sigma1(w[round+1 ]) + w[round+12] + sigma0(w[round+4]);
```

이때, 2개의 종속성, 즉 w[round]와 w[round+1]가 존재한다. SHA-256의 치환 방법은, 이전과 같이, 제로 값으로 치환한 후 그 결과를 결정하여서 이루어진다. 이 방법이 아래에 설명되어 있다:

```
w[round  ] = sigma1(w[round+14]) + w[round+9 ] + sigma0(w[round+1]);
w[round+1] = sigma1(w[round+15]) + w[round+10] + sigma0(w[round+2]);
w[round+2] = sigma1(w[0      ]) + w[round+11] + sigma0(w[round+3]);
w[round+3] = sigma1(w[0      ]) + w[round+12] + sigma0(w[round+4]);
w[round+2] += sigma1(w[round]);
w[round+3] += sigma1(w[round]);
```

상기 블록의 코드를 리팩토링 하여 데이터패스의 사이즈가 4×32비트인 SIMD 아키텍처상에서 4레인 벡터 연산을 이용할 수 있다.

SHA-256 스케쥴 갱신 및 해시 갱신 균형 맞추기

상기 스케쥴 갱신 연산과 상기 해시 갱신 연산의 균형을 맞추기 위해서, 우리는, 상기 스케쥴 갱신을, 이전에 설명한 것처럼, 즉 4-웨이 벡터화를 사용하여 처리하는 것을 제안한다. 이에 따라 단일 스케쥴 갱신을 할 수 있어 이후의 해시 함수 명령어에 대해 충분한 데이터, 4×32비트 워드를 발생시킨다.

상기 벡터화 기술은, 합리적인 SIMD 구현에 있어서, 상기 제안된 SHA-1 해시 함수를 실행하는데 걸리는 실행 사이클보다 상기 스케쥴 데이터를 계산하는데 많은 실행 사이클이 걸릴 것이다.

이것에 대해서는 이유가 여러 가지다:

요소 {round+1, round+2, round+3, round+4} 및 {round+9, round+10, round+11, round+12}를 포함하는 벡터는, 1개보다 많은 벡터 레지스터에 걸쳐 있다.

요소 {round+14, round+15, 0, 0}를 포함하는 레지스터는 추출을 이용하여 구성될 필요가 있다.

시그마(sigma) 연산은 회전을 포함하고, SIMD 벡터 회전은 SIMD 명령어 세트, 예를 들면 ARM사의 개선된 SIMD에서 일반적으로 발견되지 않고 있다. 이러한 아키텍처에서의 벡터 회전은, 2개의 벡터 시프트와 하나의 OR 명령어를 필요로 한다.

상기 치환을 고려하는 결정도, 상기 레지스터들의 추출을 필요로 한다.

상기 sigma0와 sigma1 연산은, 대략 7벡터 연산으로 이루어진다.

암달의 법칙으로 인해서, SHA-256 알고리즘의 양쪽 부분은 균형을 맞추어서 속도가 보다 느린 부품이 달성 가능한 많은 속도향상을 제한하지 않도록 할 필요가 있다.

이들의 소견에 따라 SHA-256 스케쥴 갱신 함수를 가속하기 위한 다음의 SIMD 명령어가 행해지게 된다.

SHA256SU0 Vd.4S, Vn.4S

T<127:0> = Vn<31:0>:Vd<127:32>

T<127:0> = VecROR32(T, 7) XOR VecROR32(T, 18) XOR VecSHR32(T, 3)

Vd = VecADD32(T, Vd)

SHA256SU1 Vd.4S, Vn.4S, Vm.4S

T0<127:0> = Vm<31:0>:Vn<127:32>

T1<63:0> = Vm<127:64>

T1<63:0> = VecROR32(T1<63:0>,17) XOR VecROR32(T1<63:0>,19) XOR VecSHR32(T1<63:0>,10)

T3<63:0> = VecADD32(Vd<63:0>,T0<63:0>)

T1<63:0> = VecADD32(T3<63:0>, T1<63:0>)

T2<63:0> = VecROR32(T1<63:0>,17) XOR VecROR32(T1<63:0>,19) XOR VecSHR32(T1<63:0>,10)

T3<63:0> = VecADD32(Vd<127:64>,T0<127:64>)

Vd = VecADD32(T3<63:0>,T2<63:0>):T1<63:0>

이 명령어는, 상기 원형 큐에는 4개의 4×32비트 벡터 레지스터가 있다는 것을 가정한다. 그 명령어는 상기 스케쥴 확장의 이용을 방해하지 않는다.

요소들의 재배치 및 추출은, 상기 명령어 내부에서 제어된다. 그 후, 상기 마이크로 구조는, 이들 및 고정된 시프트와 회전을 배선으로서 구현하도록 선택할 수 있다.

상기 명령어는, 대부분의 마이크로 구조에서 사이클 대기시간이 짧아도 된다.

따라서, 실시예의 예시에 따라, 상기 단일 명령어 다중 데이터 처리회로는, 입력 오퍼랜드Sp[127:0]를 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sq[127:0]를 생성하는, 제1 스케쥴 명령어에 의해 제어되도록 구성된다:

T[127:0]={Sp[31:0]:Sq[127:32]};

T[127:0]=VecROR32(T[127:0],7) XOR VecROR32(T[127:0],18) XOR VecROR32(T[127:0],3); 및

Sq[127:0]=VecADD32(T[127:0]), Sq[127:0]),

여기서, VecROR32(A,B)는 A내에서 각 32비트 워드의 B비트 위치만큼의 별도의 우회전이고, VecADD32(A,B)는 A내에서 각 32비트 워드를 B내에서 대응한 32비트 워드에 별도로 가산하는 것이다.

따라서, 실시예의 예시에 따라, 상기 단일 명령어 다중 데이터 처리회로는, 제1 입력 오퍼랜드 Sp[127:0]와 제2 입력 오퍼랜드Sq[127:0]을 갖고, 아래의 단계에 의해 나타낸 것과 같은 값을 갖는 출력 오퍼랜드Sr[127:0]를 생성하는, 제2 스케쥴 갱신 명령어에 의해 제어되도록 구성된 것을 제공한다:

T0[127:0] = {Sq[31:0]:Sp[127:32]};

T1[63:0] = Sq[127:64];

T1[63:0] = VecROR32(T1[63:0], 17) XOR VecROR32(T1[63:0], 19) XOR VecROR32(T1[63:0], 10);

T3[63:0] = VecADD32(Sr[63:0], T0[63:0]);

T1[63:0] = VecADD32(T3[63:0], T1[63:0]);

T2[63:0] = VecROR32(T1[63:0], 17) XOR VecROR32(T1[63:0], 19) XOR VecROR32(T1[63:0], 10);

T3[63:0] = VecADD32(Sr[127:64], T0[127:64]); and

Sr[127:0] = {VecADD32(T3[63:0], T2[63:0]):T1[63:0]},

여기서, VecROR32(A,B)는 A내에서 각 32비트 워드의 B비트 위치만큼의 별도의 우회전이고, VecADD32(A,B)는 A내에서 각 32비트 워드를 B내에서 대응한 32비트 워드에 별도로 가산하는 것이다.

SHA-256과 SHA-512간의 차이점

SHA-512 알고리즘은, 상기 SHA-256 알고리즘과 매우 유사하다. SHA-256에 대한 지원을 설명하는 상기 절에서 개요를 나타낸 해결방법도 SHA-512에 마찬가지로 적용될 수 있고, 이때 다음과 같은 작은 차이점이 있다:

입력 데이터는, 128바이트의 블록들로 분할되고, 빅 엔디언 형태의 16×64비트 워드로서 처리된다.

SHA-512는 8×64비트 워드로 동작한다.

SHA-512는 해시 함수의 80번 반복을 필요로 한다.

상기 해시 함수와 스케쥴 갱신은, 64비트 워드로 동작하고, 상이한 고정 시프트, 회전 및 xor을 포함한다.

간결함을 기하기 위해, 우리는 SHA-512 알고리즘을 생략한다.

SHA-512, SHA-384, SHA-512/256 및 SHA-512/256 알고리즘을 대상으로 하는 명령어

SHA-256 알고리즘을 대상으로 하는 명령어의 동기는, SHA-512 알고리즘과 마찬가지다.

우리는 이들 SHA-512 알고리즘을 대상으로 하는 명령어의 가능성이 있는 실현을 열거하고 있다.

상기 SIMD 레지스터가 128비트인 경우, 해시 및 스케쥴 명령어마다 4번 반복이라고 가정한다:

SHA512H	{Qd, Qd+1}, {Qn, Qn+1}, {Vm.2D, Vm+1.2D}
SHA512H2	{Qd, Qd+1}, {Qn, Qn+1}, {Vm.2D, Vm+1.2D}
SHA512SU0	{Vd.2D, Vd+1.2D}, {Vn.2D, Vn+1.2D}
SHA512SU1	{Vd.2D, Vd+1.2D}, {Vn.2D, Vn+1.2D}, {Vm.2D, Vm+1.2D}

이때, 위의 명령어는 레지스터 피닝(pinning); 상기 마이크로 구조내에서, 한쪽의 레지스터를 지정하는 것과 제2의 레지스터를 암시하는 것을 필요로 할 것 같다. 그 명령어는, 더 이상 전형적 RISC 3개 한 벌의 형태가 되지 않을 것 같지만, 예를 들면 ARM사의 Neon 로드/스토어 다중 명령어에서, 이들 종류의 연산에 대해 우선권이 있다.

보다 넓은 SIMD 레지스터가 이용 가능한 경우, 그 명령어들의 가능한 변형은 다음을 포함한다:

SHA512H Od, On, Vm.4D

SHA512H2 Od, On, Vm.4D

SHA512SU0 Vd.4D, Vn.4D

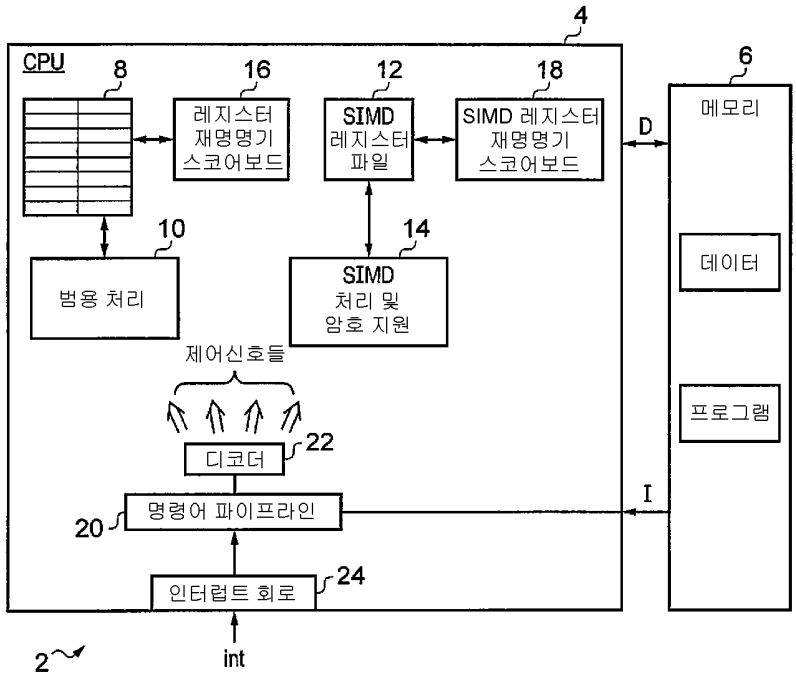
SHA512SU1 Vd.4D, Vn.4D, Vm.4D

이들도, 상기 해시 및 스케쥴 갱신 연산의 반복동안 처리하지만, 상기 보다 넓은 SIMD 레지스터로 인해 상기 3개 한 벌의 RISC형태에 적합하다.

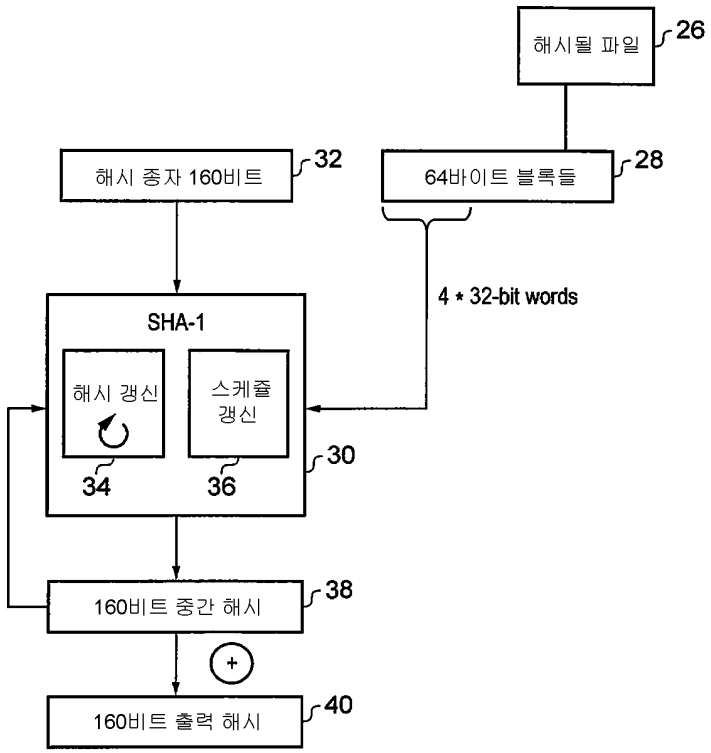
FIPS 180-4에 기재된 것처럼, 절단(truncation)을 사용한 이들 명령어도, 마찬가지로 SHA-384, SHA-512/256 및 SHA-512/224를 대상으로 할 수 있었다.

도면

도면1



도면2



도면3

제1 부분 생성