(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103930869 B
(45)授权公告日 2017.10.10

(21)申请号 201280055673.6
(22)申请日 2012.09.20
(65)同一申请的已公布的文献号
申请公布号 CN 103930869 A
(43)申请公布日 2014.07.16
(30)优先权数据
1119834.8 2011.11.17 GB
(85)PCT国际申请进入国家阶段日
2014.05.13
(86)PCT国际申请的申请数据
PCT/GB2012/052315 2012.09.20
(87)PCT国际申请的公布数据
WO2013/072657 EN 2013.05.23
(73)专利权人 ARM 有限公司
地址 英国剑桥
(72)发明人 马修·詹姆斯·霍斯内尔
理查德·罗伊·格里森思怀特
丹尼尔·克尔肖
斯图亚特·大卫·贝尔斯
(74)专利代理机构 北京东方亿思知识产权代理有限责任公司 11258
代理人 李晓冬
(51)Int.Cl.
G06F 9/30(2006.01)
G06F 9/38(2006.01)
H04L 9/08(2006.01)
(56)对比文件
CN 101520965 A,2009.09.02,全文.
审查员 梁岩

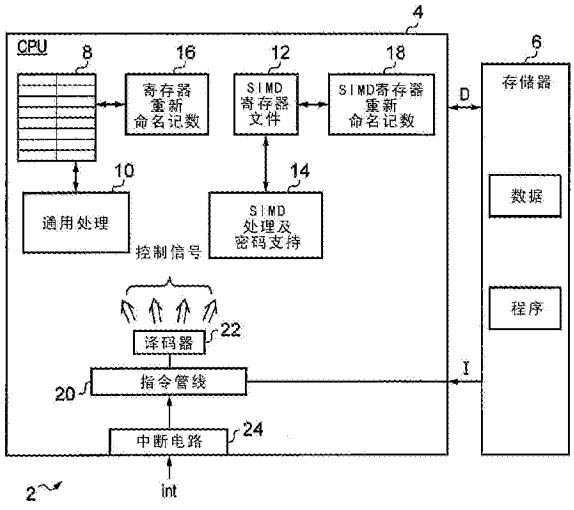
权利要求书9页 说明书24页 附图3页

(54)发明名称

密码算法中用于支持散列值生成的SIMD指令

(57)摘要

数据处理系统(2)包括单一指令多重数据寄存器文件(12)及单一指令多重处理电路(14)。单一指令多重数据处理电路(14)支持用于执行散列算法的部分的密码处理指令的执行。操作数存储在单一指令多重数据寄存器文件(12)内。支持密码的指令不遵循一般基于通道的处理且产生输出操作数,在这些输出操作数中,输出操作数的不同部分取决于在输入操作数内的多个不同元素。



CN 103930869 B

1.一种数据处理设备，该数据处理设备包含：

单一指令多重数据寄存器文件；及

单一指令多重数据处理电路，该单一指令多重数据处理电路耦接至该单一指令多重数据寄存器文件且被配置为由单一指令多重数据程序指令控制，以独立地对存储在单独通道内的单独数据元素执行处理操作，该单独通道在该单一指令多重数据寄存器文件的输入操作数寄存器内；其中

该单一指令多重数据处理电路被配置为由另外的程序指令控制，以对向量数据值执行另外的处理操作，该向量数据值包含保持在该单一指令多重数据寄存器文件的输入操作数寄存器内的数据元素序列，以产生存储在该单一指令多重数据寄存器文件的输出操作数寄存器内的输出操作数，该输出操作数具有含有值的第一部分，该值取决于在该数据元素序列内的所有数据元素；

其中该另外的程序指令为密码程序指令，该密码程序指令操作以依赖于形成该向量数据值的数据的多个字产生作为该输出操作数的输出散列值；以及

该另外的程序指令执行迭代处理操作，该迭代处理操作消耗连续数据字及至少部分中间散列值以产生该输出散列值。

2.如权利要求1所述的数据处理设备，其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者，且该向量数据值包含$Vm[32*2^N-1:0]$，其中N为正整数，该另外的处理操作产生该输出操作数$Qd_{output}[127:0]$，以具有与由以下步骤给出的值相同的值：

```
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];
```

其中OP_FUNC(B,C,D)为以下其中之一：

(((C XOR D)AND B)XOR D)；

(B XOR C XOR D)；及

(B AND C)OR((B OR C)AND D);及

ROL(P,Q)为值P左旋转Q位位置。

3.如权利要求1所述的数据处理设备,其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者,且该向量数据值包含Vm[32*2^N-1:0],其中N为正整数,该另外的处理操作产生该输出操作数Qd_{output}[127:0],以具有与由以下步骤给出的值相同的值:

```
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]};
```

其中OP_FUNC(B,C,D)为以下其中之一:

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

(B XOR C XOR D);及

(B AND C)OR((B OR C)AND D);及

ROL(P,Q)为值P左旋转Q位位置。

4.如权利要求2和3中任一项所述的数据处理设备,其中该另外的程序指令包括选择以下其中之一作为OP FUNC(B,C,D)的字段:

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

(B XOR C XOR D);及

(B AND C)OR((B OR C)AND D)。

5.如权利要求2和3中任一项所述的数据处理设备,其中该第一输入操作数Qd[127:0]及该第二输入操作数Sn[31:0]读取自在该单一指令多重数据寄存器文件内的单独寄存器。

6.如权利要求2和3中任一项所述的数据处理设备,其中该第一输入操作数Qd[127:0]及该第二输入操作数Sn[31:0]读取自在该单一指令多重数据寄存器文件内的共享寄存器。

7.如权利要求1所述的数据处理设备,其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者,且该

向量数据值包含Vm[32*2^N-1:0]，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

```
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];
```

其中Choose(B,C,D)为(((C XOR D)AND B)XOR D)，Majority(B,C,D)为((B AND C)OR((B OR C)AND D))，Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22))，Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

8.如权利要求1所述的数据处理设备，其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者，且该向量数据值包含Vm[32*2^N-1:0]，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

```
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];
```

其中Choose(B,C,D)为(((C XOR D)AND B)XOR D),Majority(B,C,D)为((B AND C)OR((B OR C)AND D)),Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22)),Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

9.如权利要求7和8中任一项所述的数据处理设备,其中该第一输入操作数Qd[127:0]及该第二输入操作数Qn[127:0]读取自在该单一指令多重数据寄存器文件内的单独寄存器。

10.如权利要求7和8中任一项所述的数据处理设备,其中该第一输入操作数Qd[127:0]及该第二输入操作数Qn[127:0]读取自在该单一指令多重数据寄存器文件内的共享寄存器。

11.如权利要求1所述的数据处理设备,其中该单一指令多重数据处理电路利用用于管理该另外的程序指令及该单一指令多重数据程序指令的处理的共同机制。

12.如权利要求11所述的数据处理设备,其中该管理处理包括管理下述中的一个或多个:

寄存器重新命名;

指令调度;

指令发出;

指令报废;及

指令中断。

13.如权利要求2和3中任一项所述的数据处理设备,其中该单一指令多重数据处理电路被配置为由旋转指令控制,该旋转指令具有输入操作数Sm[31:0]且产生具有值的输出操

作数Sd[31:0]，该值与Sm[31:0]右旋转两个位位置给出的值相同。

14.如权利要求2和3中任一项所述的数据处理设备，其中该单一指令多重数据处理电路被配置为由第一调度更新指令控制，该第一调度更新指令具有第一输入操作数Sp[127:0]及第二输入操作数Sq[127:0]且产生具有值的输出操作数Sr[127:0]，该值与通过以下步骤给出的值相同：

T[127:0]＝{Sp[63:0]:Sq[127:64]}及

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

15.如权利要求2和3中任一项所述的数据处理设备，其中该单一指令多重数据处理电路被配置为由第二调度更新指令控制，该第二调度更新指令具有输入操作数Ss[127:0]且产生具有值的输出操作数St[127:0]，该值与通过以下步骤给出的值相同：

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

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

St[127:96]＝({T[126:96]:T[127]})XOR({T[29:0]:T[31:30]})。

16.如权利要求7和8中任一项所述的数据处理设备，其中该单一指令多重数据处理电路被配置为由第一调度更新指令控制，该第一调度更新指令具有输入操作数Sp[127:0]且产生具有值的输出操作数Sq[127:0]，该值与通过以下步骤给出的值相同：

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)；及

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

其中VecROR32(A,B)为在A内的每个32位字单独右旋转B位位置，且VecADD32(A,B)为在A内的每个32位字单独增加至在B内的相应32位字。

17.如权利要求7和8中任一项所述的数据处理设备，其中该单一指令多重数据处理电路被配置为由第二调度更新指令控制，该第二调度更新指令具有第一输入操作数Sp[127:0]及第二输入操作数Sq[127:0]且产生具有值的输出操作数Sr[127:0]，该值与通过以下步骤给出的值相同：

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])；及

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

其中VecROR32(A,B)为在A内的每个32位字单独右旋转B位位置，且VecADD32(A,B)为在A内的每个32位字单独增加至在B内的相应32位字。

18.如权利要求1所述的数据处理设备，该数据处理设备进一步包含与该单一指令多重

数据寄存器文件分离的通用寄存器文件，该通用寄存器文件具有通用寄存器，该通用寄存器具有比在该单一指令多重数据寄存器文件内的寄存器位宽度低的位宽度，并且通用处理电路耦接至该通用寄存器文件，且被配置为由通用处理指令控制以对存储在该通用寄存器中的一个通用寄存器内的输入操作数执行处理操作。

19.一种处理数据的方法，该方法包含以下步骤：

将单一指令多重数据操作数存储在单一指令多重数据寄存器文件内；

在单一指令多重数据程序指令的控制下，独立地对存储在该单一指令多重数据寄存器文件的输入操作数寄存器内的单独通道内的单独数据元素执行处理操作；及

在另外的程序指令的控制下，对向量数据值执行另外的处理操作，该向量数据值包含保持在该单一指令多重数据寄存器文件的输入操作数寄存器内的数据元素序列，以产生存储在该单一指令多重数据寄存器文件的输出操作数寄存器内的输出操作数，该输出操作数具有含有值的第一部分，该值取决于在该数据元素序列内的所有数据元素；

其中该另外的程序指令为密码程序指令，该密码程序指令操作以依赖于形成该向量数据值的数据的多个字产生作为该输出操作数的输出散列值；以及

在该另外的程序指令的控制下，迭代处理操作被执行，该迭代处理操作消耗连续数据字及至少部分中间散列值以产生该输出散列值。

20.如权利要求19所述的方法，其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者，且该向量数据值包含Vm[32*2^N-1:0]，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与通过以下步骤给出的值相同的值：

```
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];
```

其中OP FUNC(B,C,D)为以下其中之一：

(((C XOR D)AND B)XOR D)；

(B XOR C XOR D);及

(B AND C)OR((B OR C)AND D);及

ROL(P,Q)为值P左旋转Q位位置。

21.如权利要求19所述的方法,其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者,且该向量数据值包含Vm[32*2^{N}-1:0],其中N为正整数,该另外的处理操作产生该输出操作数Qd_{output}[127:0],以具有与通过以下步骤给出的值相同的值:

```
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]};
```

其中OP FUNC(B,C,D)为以下其中之一:

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

(B XOR C XOR D);及

(B AND C)OR((B OR C)AND D);及

ROL(P,Q)为值P左旋转Q位位置。

22.如权利要求19所述的方法,其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者,且该向量数据值包含Vm[32*2^{N}-1:0],其中N为正整数,该另外的处理操作产生该输出操作数Qd_{output}[127:0],以具有与通过以下步骤给出的值相同的值:

```
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];

其中Choose(B,C,D)为(((C XOR D)AND B)XOR D),Majority(B,C,D)为((B AND C)OR((B OR C)AND D)),Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22)),Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

23.如权利要求19所述的方法,其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者,且该向量数据值包含$Vm[32*2^N-1:0]$,其中N为正整数,该另外的处理操作产生该输出操作数Qd_{output}[127:0],以具有与通过以下步骤给出的值相同的值:

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];

其中Choose(B,C,D)为(((C XOR D)AND B)XOR D),Majority(B,C,D)为((B AND C)OR((B OR C)AND D)),Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22)),Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

密码算法中用于支持散列值生成的SIMD指令

技术领域

[0001] 本发明关于数据处理系统领域。更具体地，本发明关于在数据处理系统内提供支持密码的指令。

背景技术

[0002] 使用数据处理系统执行密码操作是为人们已知的。这种已知密码处理操作的实例包括安全散列算法(Secure Hash Algorithm;SHA)。SHA具有各种不同已知形式，包括SHA-1、SHA-2、SHA256及SHA512。这些算法是运算密集的算法。

[0003] 一种支持这些算法的已知方法是使用通用处理器，该通用处理器利用该通用处理器的通用寄存器文件(register file)执行通用指令。此方法的一个问题在于执行这些算法必须操纵大量状态数据(通常可产生160位及160位以上的散列值)，结果是操作通常必须被分解并且每次由对数据的部分进行操作的单独程序指令的长序列执行，进而导致执行算法所需的时间量及执行算法时消耗的能量的不利增加。

[0004] 另一已知方法是提供专用支持密码的处理器，诸如密码共处理器，该密码共处理器具有专用电路，该专用电路用于执行该算法且通常通过传递待散列的数据的开始的指针且然后等待接收所得散列值而启动。此方法之问题在于：提供专用密码硬件产生了额外成本及复杂性。此外，在将专用硬件的操作与装置的其他操作整合时会产生问题，诸如中断处理、多任务等等，因为将专用密码硬件并入通常在数据处理系统内提供的机制中以使用数据处理系统处理操作的这些方面非常困难且复杂。

发明内容

[0005] 自一个方面而言，本发明提供一种数据处理设备，该数据处理设备包含：

[0006] 单一指令多重数据寄存器文件；及

[0007] 单一指令多重数据处理电路，该单一指令多重数据处理电路耦接至该单一指令多重数据寄存器文件且被配置为由单一指令多重数据程序指令控制，以独立地对存储在单独通道内的单独数据元素执行处理操作，该单独通道在该单一指令多重数据寄存器文件的输入操作数寄存器内；其中

[0008] 该单一指令多重数据处理电路被配置为由另外的程序指令控制，以对向量数据值执行另外的处理操作，该向量数据值包含保持在该单一指令多重数据寄存器文件的输入操作数寄存器内的数据元素序列，以产生存储在该单一指令多重数据寄存器文件的输出操作数寄存器内的输出操作数，该输出操作数具有含有值的第一部分，该值取决于在该数据元素序列内的所有数据元素。

[0009] 本发明技术认识到，许多数据处理系统已具备单一指令多重数据处理机制。这些单一指令多重数据处理机制通常包括单一指令多重数据寄存器文件，该单一指令多重数据寄存器文件具有能够存储且操纵大数据宽度操作数的大存储容量，这些大数据宽度操作数通常在单一指令多重数据处理中涉及。在单一指令多重数据处理中，通常在单一程序指令

控制下独立地处理数据的单独通道。例如，数据的单独通道可包含色彩像素值或其他向量值的分量值，所有这些值皆将经受相同处理操作，诸如缩放。本发明技术认识到，单一指令多重数据寄存器文件的存储能力可利用另外的程序指令重复使用，该另外的程序指令不遵循单一指令多重数据程序指令的通常形式。特定言之，通道的处理不必为独立的，且产生的输出操作数可具有含有值的第一部分，该值取决于在形成输入的向量数据值内的所有数据元素。

[0010] 在单一指令多重数据程序指令的区域外的单一指令多重寄存器文件的重复使用可应用于各种领域，诸如数据压缩及数据密码术。本技术尤其非常适合于数据密码术。

[0011] 在此上下文中，另外的程序指令可经安排以执行迭代处理操作，该迭代处理操作消耗连续的数据字及至少部分中间散列值以产生输出散列值。散列值产生通常需要操纵大量的数据及寄存器文件，同时具有存储及操纵非常长的操作数值的能力。

[0012] 另外的程序指令的一个形式为其中该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者，且该向量数据值包含Vm[Index+31:Index]，其中Index为0至2^N，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

[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]}
}
Q_output[127:0]=X[127:0];
```

[0014] 其中OP FUNC(B,C,D)为以下其中之一：

[0015] (((C XOR D)AND B)XOR D)；

[0016] (B XOR C XOR D)；及

[0017] (B AND C)OR((B OR C)AND D)；及

[0018] ROL(P,Q)为值P左旋转Q位位置。

[0019] 迭代程序指令的该形式非常适合于实施SHA-1算法。应了解，上文定义的操作能够

以伪代码形式给出，且上文定义的操作能够以各种不同硬件形式实施，熟习该领域技术的技术人员将很好地理解此点。特定言之，低电路额外负担实施可再循环值以执行迭代操作，其中较高效能实施可寻求并行执行至少部分不同迭代。

[0020] 另外的程序指令的另一形式具有自该单另一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者，且该向量数据值包含Vm[Index+31:Index]，其中Index为0至2^N，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

```
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]};
```

[0021]

[0022] 其中OP FUNC(B,C,D)为以下其中之一：

[0023] (((C XOR D)AND B)XOR D)；

[0024] (B XOR C XOR D)；及

[0025] (B AND C)OR((B OR C)AND D)；及

[0026] ROL(P,Q)为值P左旋转Q位位置。

[0027] 由OP FUNC评估的函数的选择可依赖于在另外的程序指令内的特定字段进行，或该选择可依赖于在处理待散列的数据值之当前输入区块期间已执行多少次迭代。

[0028] 在一些实施例中，单一指令多重数据寄存器文件可能不具有将所有第一输入操作数及第二输入操作数存储在单个寄存器中的能力，且因此这些输入操作数可存储于在单一指令多重数据寄存器文件内的单独寄存器内。在其他实施例中，第一输入操作数及第二输入操作数可存储在共享寄存器内，且第一输入操作数及第二输入操作数可视为单个输入操作数。

[0029] 在进一步实施例中，与上述另外的程序指令结合或代替上述另外的程序指令，本发明技术可提供对另外的程序指令的支持，该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者，且该向量

数据值包含Vm[Index+31:Index]，其中Index为0至2^N，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

[0030]

```
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],Y[63:32],Y[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]}
 }
Q_output[127:0]=X[127:0];
```

[0031] 其中Choose(B,C,D)为(((C XOR D)AND B)XOR D)，Majority(B,C,D)为((B AND C)OR((B OR C)AND D))，Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22))，Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

[0032] 以类似方式，另外的程序指令亦可具有一形式，在该形式中，另外的程序指令具有自该单另一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者，且该向量数据值包含Vm[Index+31:Index]，其中Index为0至2^N，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

[0033]

```
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],Y[63:32],Y[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];
```

[0034] 其中Choose(B,C,D)为(((C XOR D)AND B)XOR D),Majority(B,C,D)为((B AND C)OR((B OR C)AND D)),Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22)),Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

[0035] 上述两种形式的另外的程序指令特别适合于支持SHA-224算法及SHA256算法。

[0036] 用于管理另外的程序指令的处理的机制可方便地与单一指令多重数据处理电路结合。用于管理另外的处理指令的处理的机制使用单一指令多重数据指令寄存器文件,且当用于管理另外的程序指令的处理(例如中断处理、调度)的机制与单一指令多重数据处理电路的机制整合时,实施可简化。

[0037] 管理可与单一指令多重数据处理电路的程序指令整合的另外的程序指令的处理的方面包括寄存器重新命名、指令调度、指令发出、指令报废及指令中断。单一指令多重数据处理电路通常已包括管理且支持这些操作的电路组件,且另外的程序指令可相对容易地整合至该管理支持中。此举提供了以下优点:若中断在产生密码散列值中途发生,则正常中断处理机制可用以服务该中断且在服务该中断之后重新启动或继续散列计算而具有较少额外管理负担或复杂性。

[0038] 对散列算法的支持进一步通过提供旋转指令增强,该旋转指令具有输入操作数Sm[31:0]且产生具有值的输出操作数Sd[31:0],该值与Sm[31:0]右旋转两个位位置给出的值相同。

[0039] 除产生中间散列值之外还应执行的密码散列算法处理的另一方面为更新在正在

处理的文件内的数据元素的调度。应根据散列产生的工作负载平衡该调度更新，以免引入处理量的不利瓶颈。因此，本发明的一些实施例规定该单一指令多重数据处理电路被配置为由第一调度更新指令控制，该第一调度更新指令具有第一输入操作数Sp[127:0]及第二输入操作数Sq[127:0]且产生具有值的输出操作数Sr[127:0]，该值与由以下步骤给出的值相同：

[0040] T[127:0]={Sp[63:0]:Sq[127:64]}及

[0041] Sr[127:0]=T[127:0]XOR Sr[127:0]XOR Sq[127:0].

[0042] 此外，一些实施例规定该单一指令多重数据处理电路被配置为由第二调度更新指令控制，该第二调度更新指令具有输入操作数Ss[127:0]且产生具有值的输出操作数St[127:0]，该值与由以下步骤给出的值相同：

[0043] T[127:0]=St[127:0]XOR{32{0}:Ss[127:32]}；

[0044] St[95:0]＝{T[94:64]:T[95]:T[62:32]:T[63]:T[30:0]:T[31]}；及

[0045] St[127:96]＝({T[126:96]:T[127]})XOR({T[29:0]:T[31:30]}).

[0046] 上述两种形式的程序指令特别适合于支持SHA-256及SHA-224算法。

[0047] 为了帮助支持其他形式的散列算法的调度产生，根据一些实施例该单一指令多重数据处理电路被配置为由第一调度更新指令控制，该第一调度更新指令具有输入操作数Sp[127:0]且产生具有值的输出操作数Sq[127:0]，该值与由以下步骤给出的值相同：

[0048] T[127:0]={Sp[31:0]:Sq[127:32]}；

[0049] T[127:0]=VecROR32(T[127:0],7)XOR VecROR32(T[127:0],18)XOR VecROR32(T[127:0],3)；及

[0050] Sq[127:0]=VecADD32(T[127:0],Sq[127:0])，

[0051] 其中VecROR32(A,B)为在A内的每个32位字单独右旋转B位位置，且VecADD32(A,B)为在A内的每个32位字单独增加至在B内的相应32位字。

[0052] 进一步实施例另外提供该单一指令多重数据处理电路，该单一指令多重数据处理电路被配置为由第一调度更新指令控制，该第一调度更新指令具有第一输入操作数Sp[127:0]及第二输入操作数Sq[127:0]且产生具有值的输出操作数Sr[127:0]，该值与由以下步骤给出的值相同：

[0053] T0[127:0]={Sq[31:0]:Sp[127:32]}；

[0054] T1[63:0]=Sq[127:64]；

[0055] T1[63:0]=VecROR32(T1[63:0],17)XOR VecROR32(T1[63:0],19)XOR VecROR32(T1[63:0],10)；

[0056] T3[63:0]=VecADD32(Sr[63:0],T0[63:0])；

[0057] T1[63:0]=VecADD32(T3[63:0],T1[63:0])；

[0058] T2[63:0]＝VecROR32(T1[63:0],17)XOR VecROR32(T1[63:0],19)XOR VecROR32(T1[63:0],10)；

[0059] T3[63:0]＝VecADD32(Sr[127:64],T0[127:64])；及

[0060] Sr[127:0]={VecADD32(T3[63:0],T2[63:0]):T1[63:0]}，

[0061] 其中VecROR32(A,B)为在A内的每个32位字单独右旋转B位位置，且VecADD32(A,B)为在A内的每个32位字单独增加至B内的相应32位字。

[0062] 上述两种形式的程序指令特别适合于支持SHA-256算法。

[0063] 自另一方面而言，本发明提供数据处理设备，该数据处理设备包含：

[0064] 单一指令多重数据寄存器文件装置，用于存储单一指令多重数据操作数；及

[0065] 单一指令多重数据处理装置，用于在单一指令多重数据程序指令的控制下执行处理操作，该单一指令多重数据处理装置耦接至该单一指令多重数据寄存器文件装置，且对存储在该单一指令多重数据寄存器文件装置的输入操作数寄存器内的单独通道内的单独数据元素独立地执行该处理操作；其中

[0066] 该单一指令多重数据处理装置由另外的程序指令控制，以对向量数据值执行另外的处理操作，该向量数据值包含保持在该单一指令多重数据寄存器文件装置的输入操作数寄存器内的数据元素序列，以产生存储在该单一指令多重数据寄存器文件装置的输出操作数寄存器内的输出操作数，该输出操作数具有含有值的第一部分，该值取决于在该数据元素序列内的所有数据元素。

[0067] 自进一步方面而言，本发明提供一种处理数据的方法，该方法包含以下步骤：

[0068] 将单一指令多重数据操作数存储在单一指令多重数据寄存器文件内；

[0069] 在单一指令多重数据程序指令的控制下，独立地对存储在该单一指令多重数据寄存器文件的输入操作数寄存器内的单独通道内的单独数据元素执行处理操作；及

[0070] 在另外的程序指令的控制下，对向量数据值执行另外的处理操作，该向量数据值包含保持在该单一指令多重数据寄存器文件的输入操作数寄存器内的数据元素序列，以产生存储在该单一指令多重数据寄存器文件的输出操作数寄存器内的输出操作数，该输出操作数具有含有值的第一部分，该值取决于在该数据元素序列内的所有数据元素。

[0071] 本发明的另一方面提供虚拟机实施，该虚拟机实施提供在通用计算机上的执行环境，该执行环境许可上文详述的程序指令如同这些程序指令在上文详述的数据处理设备上执行一样执行。在本文中涵盖本技术的这些虚拟机实施。

附图说明

[0072] 下面通过举例，并参照附图描述本发明的实施例，附图中：

[0073] 图1示意性示出包括单一指令多重数据寄存器文件及单一指令多重数据处理电路的数据处理设备，该单一指令多重数据处理电路包括对执行密码处理指令的支持；

[0074] 图2示意性示出在散列算法的一示例性形式内的数据流；及

[0075] 图3示意性示出另外的处理指令如何不遵循与单一指令多重数据处理电路相关联的一般的基于通道的处理。

具体实施方式

[0076] 图1示意性示出中央处理单元4形式的数据处理设备2，该中央处理单元4耦接至存储器6，该存储器6存储待操纵的数据及待执行的程序指令。中央处理单元4包括通用寄存器文件8、通用处理电路10、单一指令多重数据寄存器文件12及单一指令多重数据处理电路14。通用寄存器文件8通常含有低位宽度通用寄存器(例如32位或64位)，诸如具有由英国剑桥的ARM有限公司生产的处理器的通用寄存器文件支持的形式的寄存器。单一指令多重数据寄存器文件12通常包括更大的寄存器，且在单一指令多重数据寄存器文件12之内的数据

存储可取决于所利用的寄存器大小说明符以不同方式划分以形成不同寄存器。单一指令多重数据寄存器文件12的形式可为由英国剑桥的ARM有限公司生产的处理器的一些实施中支持的Neon寄存器文件的形式。

[0077] 通用寄存器重新命名及记数电路(score boarding circuitry)16与通用寄存器文件10相关联，且单一指令多重数据寄存器重新命名及记数电路18与单一指令多重数据寄存器文件12相关联。寄存器重新命名及记数自身是本领域技术人员所熟知的已知技术且将不在本文中进一步描述。与提供寄存器重新命名及记数用于一般单一指令多重数据处理指令的方式相同，寄存器重新命名及记数可应用于在下文中进一步描述的密码处理指令的支持中所利用的寄存器。因此，已提供用于支持寄存器重新命名、指令调度、指令发出、指令报废及指令中断的机制可由支持密码的程序指令重新使用，且相应地，这些支持密码的指令的操作可较好地与中央处理单元4的整体操作整合。

[0078] 程序指令I接收自存储器6且传递至指令管线(instruction pipeline)20。指令译码器22将程序指令译码且产生控制信号，该控制信号控制寄存器文件8、12及处理电路10、14，以及在中央处理单元4内的其他组件的操作。中断电路24响应于外部产生的中断信号int以中断当前由中央处理单元4执行的处理，且开始执行为本技术领域技术人员所熟知的中断处理代码。应了解，中央处理单元4通常将包括许多附加电路组件，且为了清楚起见，已将这些附加电路组件从第1图中省略。

[0079] 图2示意性图示了散列产生算法的一种形式内的数据流。待散列的文件26被分割成64字节区块28，该64字节区块28经进一步分割成作为一个输入提供至散列算法30的四个32位字的输入向量。在散列操作开始时还提供散列种子值32。散列操作使用两个主程序回路，该两个主程序回路分别负责散列更新34及调度更新36。这些主回路通过在单一指令多重数据处理电路14中提供支持两个回路的专用程序指令来平衡。中间散列值38由散列更新回路34产生且在散列算法持续处理输入数据的区块28时被反馈。当已处理区块28(例如，在SHA-1情况下经历80个散列更新)时，然后通过将当前中间散列值38添加至输出散列值40中而更新输出散列值40。重复该过程直至全部文件26已用完为止。此举总体上产生文件26的最终散列值。

[0080] 散列更新回路34将被执行多次，且散列更新回路34自身执行指令，这些指令中的每个指令具有其自身的指令内部迭代，下文将对其进行描述。执行调度更新36以便平衡散列更新。调度更新可经向量化以改良效能，下文将对其进行描述。

[0081] 图3示意性图示根据本发明技术的另外的处理指令如何接收包含多个数据元素的向量数据值42。然后，支持密码的指令对该向量数据值42执行处理操作以产生具有第一部分44的输出操作数，该第一部分44取决于向量数据值42的第一数据元素及在向量数据值内的两个或两个以上另外的数据元素。该行为与典型单一指令多重数据程序指令形成对比，在该典型单一指令多重数据程序指令中处理操作是基于通道的且在不同通道内的数据值之间存在有限的互动。

[0082] 本技术之一实施是一组指令，该组指令将两个算法作为目标，亦即，算法SHA-1及SHA-256。指令亦对SHA-224算法有利，SHA-224算法需要与SHA-256相同的操作。SHA算法为由国家标准技术局(National Institute of Standards and Technology;NIST)规定的安全散列算法系列。这些算法的规范可公开得到。这些算法通常用于验证数字系统内的数据。

[0083] 我们从描述SHA算法的高阶操作且包括用于SHA-1及SHA-256算法的伪码开始。

[0084] SHA算法的高阶操作(已知;FIPS180-4)

[0085] 算法中的每个算法处理64字节数据,且产生散列摘要(digest);在SHA-1的情况下,为160位的长度,且在SHA-256的情况下,为256位的长度。长度大于64字节的数据流被分为64字节区块。在流或区块长度小于64字节的情况下,可按照在美国联邦信息处理标准(Federal Information Processing Standard;FIPS)180-4中所规定将区块填充至64字节。除非另有说明,否则算法的以下描述假设字为32位无符号整数值。假设字由来自大端(big endian)形式的64字节区块的数据的4个相连字节组成。

[0086] 两个算法均通过初始化工作的散列摘要开始。若数据区块为给定数据流中的第一个,则将散列摘要初始化至固定种子值。若区块为数据流的延续,则将散列摘要初始化至由先前区块计算的散列摘要。在FIPS180-4中规定种子值。

[0087] 算法使用调度更新操作扩展区块。对于SHA-1,将区块自初始16数据字扩展为80字;对于SHA-256,扩展为64字。调度更新操作使用固定逻辑异或(xor)将来自调度的四个字结合,并且将该四个字移位且旋转,以在扩展调度中产生下一个字。初始16个字在扩展调度中保持不变。

[0088] 然后,扩展调度中的每个字具有添加至该字的键值。在SHA-1中,存在4个键常数,每个键常数应用于来自扩展区块的一组20个字。在SHA-256中,存在64个键常数,扩展区块的每个字有一个键常数。在FIPS180-4中定义键常数。在已扩展区块且已添加键常数之后,使用散列更新函数处理每个字,该散列更新函数经由一系列固定逻辑异或并入该字,并且移位且旋转至散列摘要中。

[0089] 最终,在已使用散列更新函数处理来自扩展区块的每个字之后,将散列摘要添加至先前散列摘要值。

[0090] 如FIPS180-4中所规定,调度可实施为80/64字(SHA-1/SHA-256)的集合或实施为16字的循环队列。

[0091] 为完整起见,假定循环队列的SHA-1及SHA-256的伪码算法如下。

[0092] SHA-1算法伪码

[0093] uint32w[0:15]=164-bytes(big-endian)input[];

[0094] uint32wk[0:15]=w[0:15]+k[0:15];

[0095] uint32a:e=io->hashes[0:4];

[0096] for round=0:63{

[0097] hash_update(round,wk[round]);

[0098] w[round]=scheduleupdate(round,w);

[0099] wk[round]=w[round]+k[round];

[0100] }

[0101] for round=64:79

[0102] hash_update(round,w[round]);

[0103] io->hashes[0:4]+=a:e;

[0104] SHA-1散列更新代码如下:

[0105] hash_update(int round,uint32wk){

[0106] e+=FN(round,b,c,d)+ROL(a,5)+wk;

[0107] b=ROL(b,30);

[0108] rotate(a,b,c,d,e)to(e,a,b,c,d)

[0109] }

[0110] 其中：

[0111] if round<20,FN=choose(b,c,d);

[0112] else if round<40,FN=parity(b,c,d);

[0113] else if round<60,FN=maj ority(b,c,d);

[0114] else FN=Parity(b,c,d);

[0115] choose(b,c,d)=(((c^d)&b)^d)

[0116] parity(b,c,d)=(b^c^d)

[0117] majority(b,c,d)=(b&c)|((b|c)&d)

[0118] SHA-1调度更新代码如下：

[0119] uint32schedule_update(int round,uint32*w){

[0120] retum ROR(w[round-3]^W[round-8]^W[round-14]^W[round-16],31);

[0121] }

[0122] SHA-256算法伪码

[0123] uint32a:h=io->hashes[0:7];

[0124] uint32w[0:15]=164-bytes(big-endian)input[0:63];

[0125] uint32wk[0:15]=w[0:15]+k[0:15];

[0126] for round=0:47{

[0127] hash_update(wk[round]);

[0128] w[round]=schedule_update(round,w);

[0129] wk[round]=w[round]+k[round];

[0130] }

[0131] for round=48:63

[0132] hash_update(wk[round]);

[0133] io->hashes[0:7]+=a:h;

[0134] SHA-256散列更新代码如下：

[0135] hash_update(uint32wk){

[0136] t=h+Sigma1(e)+Choose(e,f,g)+wk;

[0137] d+=t;

[0138] h=t+Sigma0(a)+Majority(a,b,c);

[0139] rotate(a,b,c,d,e,f,g,h)to(h,a,b,c,d,e,f,g);

[0140] }

[0141] 其中：

[0142] Sigma0(x)=ror(x,2)^ror(x,13)^ror(x,22);

[0143] Sigma1(x)=ror(x,6)^ror(x,11)^ror(x,25);

[0144] Choose(b,c,d)=(((c^d)&b)^d)

[0145] Majority(b,c,d)=((b&c)|((b|c)&d)

[0146] 同样地,SHA-256调度更新伪码如下:

[0147] uint32schedule_update(int round,uint32*w){

[0148] return w[round]+sigma1(w[round-2])+w[round-7]+sigma0(w[round-15]);

[0149] }

[0150] 其中:

[0151] sigma0(x)=ror(x,7)^ror(x,18)^shr(x,3);

[0152] sigma1(x)=ror(x,17)^ror(x,19)^shr(x,10);

[0153] SHA算法工作状态(可源自FIPS180-4规范)

[0154] 约束经采用以加速SHA算法的方法的SHA算法的一方面为处理数据区块所需要的工作状态量(如先前所述)。单一指令多重数据寄存器文件保持且操纵此状态的能力解决该约束。

[0155] 下表概括对于SHA-1及SHA-256的状态要求。

[0156] SHA-1状态

[0157]

初始/先前散列摘要	5x32 位字
工作散列摘要	5x32 位字
调度	16x32 位字
键常数	4x32 位字

[0158] SHA-256状态

[0159]

初始/先前散列摘要	8x32 位字
工作散列摘要	8x32 位字
调度	16x32 位字
键常数	64x32 位字

[0160]

[0161] 使用SHA-1或SHA-256算法中任一者构建能够处理数据区块的专用SHA单元(例如,共处理器)需要固定目的状态的投入。该状态无法轻易地由RISC微处理器上的其他操作使用。

[0162] 将SHA算法分成三元形式RISC指令

[0163] 为了避免固定目的状态,我们已通过可于RISC微处理器上处理算法的方法将算法分解,该RISC微处理器监视三元指令形式且使用单一指令多重数据处理电路及寄存器文件。

[0164] RISC三元形式的典型约束为三个寄存器中仅一个寄存器被定义为目的地。然而,目的地可用为源。

[0165] 我们使用SIMD寄存器以便比使用通用寄存器每指令可处理更多数据。

[0166] 通过监视三元指令形式,指令能够使用为现代微处理器所常见的重新命名、调度、发出、结果及报废逻辑。

[0167] 由于所有状态及相依性由指令定义,所以处理乱序执行、中断及推测(speculation)的管线机制仍然有效;不需要附加控制逻辑即可以保持所提议指令的正确

执行。

[0168] SHA-1散列更新指令

[0169] 先前所述的SHA-1散列更新函数将32位字并入160位散列摘要中。函数由固定移位、固定逻辑异或/逻辑和(and)/逻辑或(or)及固定旋转组成。

[0170] hash_update(int round,uint32wk){

[0171] e+=FN(round,b,c,d)+ROL(a,5)+wk;

[0172] b=ROL(b,30);

[0173] rotate(a,b,c,d,e)to(e,a,b,c,d)

[0174] }

[0175] 其中:

[0176] if round<20,FN=choose(b,c,d);

[0177] else if round<40,FN=parity(b,c,d);

[0178] else if round<60,FN=majority(b,c,d);

[0179] else FN=Parity(b,c,d);

[0180] choose(b,c,d)=(((c^d)&b)^d)

[0181] parity(b,c,d)=(b^c^d)

[0182] majority(b,c,d)=(b&c)|((b|c)&d)

[0183] SHA-1散列摘要为160位,且因此对整个摘要加上32位字进行的操作不可能为32位通用三元RISC形式,且这些操作将需要很大努力才能实现为64位通用三元RISC形式;将需要更多内务处理才能将32位数据值插入第三64位操作数的高32位中。

[0184] 为此,该示例性技术将SHA-1散列函数映射至一组四个高阶SIMD指令;该四个高阶SIMD指令为SHA1C、SHA1P、SHA1M及SHA1H。

[0185] SHA1C Qd,Sn,Vm.4S[OP=C,OP_FUNC=choose]

[0186] SHA1P Qd,Sn,Vm.4S[OP=P,OP_FUNC=parity]

[0187] SHA1M Qd,Sn,Vm.4S[OP=M,OP_FUNC=majority]

[0188] SHA1H Sd,Sn

[0189] 指令SHA1C、SHA1P及SHA1M采用三个操作数。Qd保持摘要散列的前4个32位字,其中Sn保持第5个。第三操作数Vm为向量,该向量在初始实施例中保持四个32位字。这允许待由指令处理的散列更新函数的4个迭代。伪码定义这些指令的操作如下文所述。应了解,根据伪码定义指令的操作为本领域技术人员所熟知,且一旦已定义伪码,则进行(执行)由伪码定义的指令的电路的实现是常规任务。

[0190] SHA1<OP>Qd,Sn,Vm.4S

[0191] X=Qd;

[0192] Y=Sn;

[0193] For(i=0to3)

[0194] {

[0195] Index =(i*32);

[0196] t1<31:0>=OP_FUNC(X<63:32>,X<95:64>,X<127:96>);

[0197] Y =Y+ROL(X<31:0>,5)+t1<31:0>+Vm<(index+31):index>;

[0198] X<63:32>=ROL(X<63:32>,30);

[0199] //Rotate

[0200] t2<31:0>=Y;

[0201] Y =X<127:96>;

[0202] X<127:0>={X<95:0>:t2<31:0>};

[0203] }

[0204] Qd=X;

[0205] 相应地,根据示例性实施例,单一指令多重数据处理电路经配置以由另外的程序指令控制,该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者,且该向量数据值包含Vm[Index+31:Index],其中Index为0至2^N,其中N为正整数,该另外的处理操作产生该输出操作数Qd_{output}[127:0],以具有与由以下步骤给出的值相同的值:

[0206]

```
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];
```

[0207] 其中OP FUNC(B,C,D)为以下其中之一:

[0208] (((C XOR D)AND B)XOR D);

[0209] (B XOR C XOR D);及

[0210] (B AND C)OR((B OR C)AND D);及

[0211] ROL(P,Q)为值P左旋转Q位位置。

[0212] 这些指令的另一实现可涉及用于在choose()函数、parity()函数及majority()函数之间进行选择的选择。

[0213] SHA1HASH Qd,Sn,Vm.4S,#OP//#OP其中#1选择C,#2选择P,#3选择M。

[0214] RISC指令形式的约束在于仅散列摘要的前4个字可由SHA1C、SHA1P及SHA1M指令返

回至128位寄存器Qd中。因此，建议指令SHA1H返回散列摘要的第5个字。

[0215] 在首次实现中，SHA1H可实施为：

[0216] SHA1H Sd,Sn

[0217] Sd＝ROR(Sn,2)；

[0218] 此举遵循以下经验：在四次迭代后第5散列摘要值为Qd[0]初始值的旋转。

[0219] SHA1散列更新指令变体

[0220] SHA1C指令、SHA1P指令及SHA1M指令的变体可由本发明技术的其他变体扩展以允许Vm.8S或Vm.16S操作数。这些变体包括在本发明技术之内。这将允许在单一指令内处理散列更新函数的8次及16次迭代。亦即，仍将需要Vm.4S变体，因为每20次迭代之后散列更新函数需要变化。

[0221] 作为一实例，该SHA1＜OP＞Vm.8S变体：

[0222] SHA1＜OP＞Qd,Sn,Vm.8S

[0223] X＝Qd；

[0224] Y＝Sn；

[0225] for(i＝0to7)

[0226] {

[0227] Index＝(i*32)；

[0228] t1＜31:0＞＝OP_FUNC(X＜63:32＞,X＜95:64＞,X＜127:96＞)；

[0229] Y＝Y+ROL(X＜31:0＞,5)+t1＜31:0＞+Vm＜(index+31):index＞；

[0230] X＜63:32＞＝ROL(X＜63:32＞,30)；

[0231] //Rotate

[0232] t2＜31:0＞＝Y；

[0233] Y＝X＜127:96＞；

[0234] X＜127:0＞＝{X＜95:0＞:t2＜31:0＞}；

[0235] }

[0236] Qd＝X；

[0237] 操作过8次迭代及16次迭代(Vm.8S及Vm.16S)的变体将另外需要SHA1C2指令、SHA1P2指令及SHA1M2指令。这些指令将在8次或16次迭代之后为散列摘要中的第5个字产生适当值。这些新的指令将以与SHA1C指令、SHA1P指令及SHA1M指令类似的方式实施，但是在Qd寄存器中返回第5个散列摘要字，例如：

[0238] SHA1＜OP＞2Qd,Sn,Vm.8S

[0239] X＝Qd；

[0240] Y＝Sn；

[0241] for(i＝0to3)

[0242] {

[0243] Index＝(i*32)；

[0244] t1＜31:0＞＝OP_FUNC(X＜63:32＞,X＜95:64＞,X＜127:96＞)；

[0245] Y＝Y+ROL(X＜31:0＞,5)+t1＜31:0＞+Vm＜(index+31):index＞；

[0246] X＜63:32＞＝ROL(X＜63:32＞,30)；

[0247] //Rotate

[0248] t2＜31:0＞＝Y;

[0249] Y＝X＜127:96＞;

[0250] X＜127:0＞＝{X＜95:0＞:t2＜31:0＞};

[0251] }

[0252] Qd＝{0:Y＜31:0＞};

[0253] 相应地,根据示例性实施例,单一指令多重数据处理电路经配置以由另外的程序指令控制,该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Sn[31:0]两者,且该向量数据值包含Vm[Index+31:Index],其中Index为0至2^N,其中N为正整数,该另外的处理操作产生该输出操作数Qd_{output}[127:0],以具有与由以下步骤给出的值相同的值:

```
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]};
```

[0254]

[0255] 其中OP FUNC(B,C,D)为以下其中之一:

[0256] (((C XOR D)AND B)XOR D);

[0257] (B XOR C XOR D);及

[0258] (B AND C)OR((B OR C)AND D);及

[0259] ROL(P,Q)为值P左旋转Q位位置。

[0260] 若较宽SIMD数据路径可用,可实现指令的其他变体,该其他变体将整个散列摘要返回成为八位字(8x32位):

[0261] SHA1＜OP＞Od,Vn.4S

[0262] SHA1＜OP＞Od,Vn.8S

[0263] 这些指令将处理散列函数的4次及8次迭代。

[0264] SHA1散列更新微架构选项

[0265] 对于这些指令的微架构实施存在各种选项：

[0266] 这些指令的高效能实现可选择增建一些迭代逻辑且执行更多并行化执行。

[0267] 微架构可选择使用多循环级以减少暂时管线状态，且因此减少功率消耗。

[0268] 以进位储存形式进行中间架构。

[0269] 在更广泛的变体中，在可能需要显式SHA1＜OP＞2指令的情况下，检测SHA1＜OP＞2操作何时遵循相应SHA1＜OP＞函数是有可能的。在这些情况下，防止第二计算且仅转发来自数据路径的结果应是可能的。这将在管线中需要一些暂时状态。

[0270] SHA1调度更新指令

[0271] 实现自SHA-1算法的加速需要在散列更新函数与调度更新函数之间的平衡。

[0272] 先前所述的SHA-1调度更新函数将来自数据调度的四个32位字结合为单一所得字，该单一所得字扩展调度，或在循环队列的情况下，该单一所得字将调度中的字重写。

[0273] 调度更新操作由逻辑异或及固定旋转组成。

[0274] uint32schedule_update(int round,uint32*w){

[0275] return ROR(w[round-3]^w[round+8]^W[round-14]^w[round-16],31)；

[0276] }

[0277] 或在循环队列形式中：

[0278] Void schedule_update(int round,uint32w[0..15]){

[0279] w[round]=ROR(w[round+13$_{mod16}$]^w[round+8$_{mod16}$]^W[round+2$_{mod16}$]^w[round],31)；

[0280] }

[0281] 操作需要四个输入值，该四个输入值中的一个值具有破坏性。这不符合通用32-三元RISC格式。

[0282] 调度更新指令可由ARM高阶SIMD架构提供。

[0283] 为了避免存储器负载及存储器存储，我们选择实施有效执行在FIPS180-4中描述的循环队列形式的调度更新的指令。

[0284] 为了完整起见，我们包括用于调度更新的向量化方法。

[0285] SHA-1调度更新向量化及替代

[0286] 这遵循w[round]、w[round+1$_{mod16}$]及w[round+2$_{mod16}$]可以平行处理的经验。在计算w[round+3$_{mod16}$]时具有对w[round]的相依性，该相依性防止直接路由到四向向量化。

[0287] w[round]=ROR(w[round+13]^w[round+8]^W[round+2]^w[round]),31)；

[0288] w[round+1]＝ROR(w[round+14]^w[round+9]^W[round+3]^w[round+1]),31)；

[0289] w[round+2]=ROR(w[round+15]^W[round+10]^W[round+4]^W[round+2]),31)；

[0290] w[round+3]=ROR(w[round]^W[round+11]^W[round+5]^W[round+3]),31)；

[0291] 该限制可通过在w[round+3$_{mod16}$]的计算中用零替代值w[round]，且通过用附加逻辑异或及旋转步骤修复该结果来克服；该情况说明如下。

[0292] w[round]=ROR(w[round+13]^W[round+8]^W[round+2]^W[round]),31)；

[0293] w[round+1]=ROR(w[round+14]^W[round+9]^W[round+3]^W[round+1]),31)；

[0294] w[round+2]＝ROR(w[round+15]^W[round+10]^W[round+4]^W[round+2]),31)；

[0295] w[round+3]＝ROR(0^w[round+11]^w[round+5]^w[round+3]),31)；

[0296] w[round+3]＝w[round+3]^ROR(w[round],31)；

[0297] 可将上述代码区块重新因素分解以对具有4x32位的数据路径大小的SIMD架构利用4通道向量操作。

[0298] SHA-1调度更新及散列更新平衡

[0299] 为了使调度更新操作与散列更新操作平衡，可如先前所述处理调度更新，亦即，使用四向向量化处理调度更新。这允许单个调度更新为后续散列函数指令产生足够的4x32位字数据。在合理的SIMD实施中，与用以执行建议的SHA-1散列函数的技术相比，向量化技术将更多执行循环以计算调度数据。

[0300] 对此存在数个原因：

[0301] 含有元素{round+2,round+3,round+4,round+5}的向量可能将横跨两个向量寄存器。

[0302] 含有元素{round+13,round+14,round+15,0}的向量将需要自一个向量寄存器及零向量中提取。

[0303] SIMD向量旋转在例如ARM高阶SIMD等的SIMD指令集中并不常见。因此向量旋转需要两个向量偏移及or指令。

[0304] 归因于Amdahl定律，应平衡SHA-1算法的两个部分，否则较慢部分将限制可实现的加速量。

[0305] 该观察产生用于加速SHA-1调度更新函数的以下SIMD指令。

[0306] SHA1SU0Vd.4S,Vn.4S,Vm.4S

[0307] T＜127:0＞＝Vn＜63:0＞:Vd＜127:64＞

[0308] Vd＝T XORVd XORVm

[0309] SHA1SU1Vd.4S,Vn.4S

[0310] T＜127:0＞＝Vd XOR{32{0}:Vn＜127:32＞}；

[0311] Vd＜95:0＞＝T＜94:64＞:T＜95＞:T＜62:32＞:T＜63＞:T＜30:0＞:T＜31＞:

[0312] Vd＜127:96＞＝(T＜126:96＞:T＜127＞)XOR(T＜29:0＞:T＜31:30＞)；

[0313] 指令假设循环队列常驻于四个4x32位向量寄存器中。

[0314] 在指令内部完成元素的重新排序。这有效地使元素的重新排序不受拘束，这些元素正好为微架构中的线路。

[0315] 固定旋转亦仅为线路。

[0316] 在大部分微架构中，平衡指令且指令可具有极低的循环等待时间(latency)；这些指令包含串行及布线中的两个逻辑异或。

[0317] 因此，根据示例性实施例，该单一指令多重数据处理电路经配置以由第一调度指令控制，该第一调度指令具有第一输入操作数Sp[127:0]及第二输入操作数Sq[127:0]且产生输出操作数Sr[127:0]，该输出操作数的值与由以下步骤给出的值相同：

[0318] T[127:0]={Sp[63:0]:Sq[127:64]}及

[0319] Sr[127:0]=T[127:0]XOR Sr[127:0]XOR Sq[127:0].

[0320] 将SHA-2算法作为目标指令

[0321] 在针对SHA-1算法提出的指令的论述中所概括的许多特征可同样地适用于SHA-2算法。本节将描述针对SHA-2算法提出的指令中的差异。

[0322] SHA-2散列更新指令

[0323] 由于针对SHA-1概括的原因，SHA-2散列更新函数被两个散列更新指令作为目标。

[0324] SHA-2算法的工作散列摘要为256位或者512位。以下聚焦于算法SHA-256及SHA-224，该算法SHA-256及SHA-224具有256位工作散列，因为这些算法包含于本发明的初始实现中。在稍后的部分论述本发明技术如何应用于SHA-512、SHA-384、SHA-512/256及SHA-512/224。

[0325] SHA-256散列更新指令

[0326] SHA-256(及SHA-224)的工作散列摘要为256位长。在具有寄存器宽度为128位的SIMD架构中，对散列摘要的任何操作的结果需要两个指令；一个指令返回前4x32位字，且第二指令返回剩余4x32位字。

[0327] 不同于SHA-1，SHA-2散列更新函数是固定的且在给定数目迭代之后不变化，因此，我们仅需要两个指令。

```
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>);
```

[0328]

```
    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;
```

[0329] 相应地，根据示例性实施例，单一指令多重数据处理电路经配置以由另外的程序指令控制，该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者，且该向量数据值包含Vm[Index+31:Index]，其中Index为0至2^N，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

[0330]

```
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],Y[63:32],Y[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];
```

[0331] 其中Choose(B,C,D)为(((C XOR D)AND B)XOR D),Majority(B,C,D)为((B AND C)OR((B OR C)AND D)),Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22)),Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

[0332] SHA256H2Qd,Qn,Vm.4S

[0333] X＝Qn;

[0334] Y＝Qd;

[0335] for(i＝0to3)

```
        {
            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>;
```

[0336]

```
            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;
```

[0337] 相应地，根据示例性实施例，单一指令多重数据处理电路经配置以由另外的程序指令控制，该另外的程序指令具有自该单一指令多重数据寄存器文件读取的第一输入操作数Qd[127:0]及第二输入操作数Qn[127:0]两者，且该向量数据值包含Vm[Index+31:Index]，其中Index为0至2^N，其中N为正整数，该另外的处理操作产生该输出操作数Qd_{output}[127:0]，以具有与由以下步骤给出的值相同的值：

[0338]

```
    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],Y[63:32],Y[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];
```

[0339]
```
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];
```

[0340] 其中Choose(B,C,D)为(((C XOR D)AND B)XOR D),Majority(B,C,D)为((B AND C)OR((B OR C)AND D)),Sigma0(B)为(ROR(B,2)XOR ROR(B,13)XOR ROR(B,22)),Sigma1(B)为(ROR(B,6)XOR ROR(B,11)XOR ROR(B,25))且ROR(P,Q)为值P右旋转Q位位置。

[0341] SHA256H预期散列摘要的前4x32位字在Qd中,剩余4x32位字在Qn中且调度数据的4x32位字在Vm.4S中。

[0342] SHA256H2预期散列摘要的第二4x32位字在Qd中,前4x32位字在Qn中且调度数据的4x32位字在Vm.4S中。

[0343] 请注意,因为SHA256H破坏散列摘要的前4x32位字,所以在执行SHA256H之前必须进行复制以便正确值可被传递至Qn中的SHA256H2。

[0344] SHA-256散列更新指令变体

[0345] 如先前对于SHA-1散列更新指令所概述,用于较宽向量SIMD的SHA-256指令的变体可包括以下各项:

[0346] SHA256(H|H2)Qd,Qn,Vm.8S

[0347] SHA256(H|H2)Qd,Qn,Vm.16S

[0348] 这些指令将分别处理散列更新函数的8次及16次迭代。较宽的SIMD数据路径亦可允许:

[0349] SHA256H Od,Vm.8S

[0350] SHA256H Od,Vm.16S

[0351] 在Od为256位宽寄存器的情况下,不必提供SHA256H2操作,整个散列摘要将适合在向量寄存器中。

[0352] SHA-256调度更新

[0353] 如先前所概述,对于SHA-1,实现来自SHA-256算法的加速需要在散列更新函数与调度更新函数之间的平衡。

[0354] SHA-256调度更新函数将来自数据调度的四个32位字结合为单一所得字,该单一所得字扩展调度,或在循环队列的情况下,该单一所得字将调度中的字重写。

[0355] 调度更新操作由逻辑异或、固定移位及固定旋转(已知)组成。

[0356] uint32schedule_update(int round,uint32*w){

[0357] retum w[round]+sigma1(w[round-2])+w[round-7]+sigma0(w[round-15]);

[0358] }

[0359] 其中:

[0360] sigma0(x)=ror(x,7)^ror(x,18)^shr(x,3);

[0361] sigma1(x)＝ror(x,17)^ror(x,19)^shr(x,10)；

[0362] 这亦可以循环队列表达(已知)：

[0363] void schedule_update(int round,uint32*w){

[0364] w[round]＝sigma1(w[round+14$_{mod16}$])+w[round+9$_{mod16}$]+sigma0(w[round+1$_{mod16}$])；

[0365] }

[0366] SHA-256调度更新向量化及替代

[0367] SHA-256调度更新函数亦可以适合于4向SIMD的方式被向量化。

[0368] w[round]=sigma1(w[round+14])+w[round+9]+sigma0(w[round+1])；

[0369] w[round+1]=sigma1(w[round+15])+w[round+10]+sigma0(w[round+2])；

[0370] w[round+2]=sigma1(w[round])+w[round+11]+sigma0(w[round+3])；

[0371] w[round+3]=sigma1(w[round+1])+w[round+12]+sigma0(w[round+4])；

[0372] 请注意,存在两个相依性,亦即w[round]及w[round+1]。用于SHA-256的替代法通过代入零值且然后决定结果而像以前一样起作用。该方法说明如下：

[0373] w[round]＝sigma1(w[round+14])+w[round+9]+sigma0(w[round+1])；

[0374] w[round+1]＝sigma1(w[round+15])+w[round+10]+sigma0(w[round+2])；

[0375] w[round+2]＝sigma1(w[0])+w[round+11]+sigma0(w[round+3])；

[0376] w[round+3]=sigma1(w[0])+w[round+12]+sigma0(w[round+4])；

[0377] w[round+2]+＝sigma1(w[round])；

[0378] w[round+3]+＝sigma1(w[round])；

[0379] 可将上述程序代码区块重新因素分解以对具有4x32位的数据路径大小的SIMD架构利用4通道向量操作。

[0380] SHA-256调度更新及散列更新平衡

[0381] 为了使调度更新操作与散列更新操作平衡,我们提出如先前所述处理调度更新,亦即,使用四向向量化处理调度更新。这允许单个调度更新为后续散列函数指令产生足够的4x32位字的数据。

[0382] 在合理的SIMD实施中,与用以执行建议的SHA-1散列函数的技术相比,向量化技术将更多执行循环以计算调度数据。

[0383] 对此存在数个原因：

[0384] 含有元素{round+1,round+2,round+3,round+4}及{round+9,round+10,round+11,round+12}的向量将跨越多于一个向量寄存器。

[0385] 含有{round+14,round+15,0,0}的寄存器将需要使用提取组成。

[0386] sigma操作含有旋转,且SIMD向量旋转在例如ARM高阶SIMD的SIMD指令集中不常见。在这些架构中的向量旋转需要两个向量位移及OR指令。

[0387] 考虑替代的安排亦将需要提取寄存器。

[0388] sigma0及sigma1操作由大约7个向量操作组成。

[0389] 归因于Amdahl定律,需要平衡SHA-256算法之两部分以免较慢部分限制可实现的加速量。

[0390] 这些观察产生用于加速SHA-256调度更新函数的以下SIMD指令。

[0391] SHA256SU0Vd.4S,Vn.4S

[0392] T＜127:0＞＝Vn＜31:0＞:Vd＜127:32＞

[0393] T＜127:0＞＝VecROR32(T,7)XOR VecROR32(T,18)XOR VecSHR32(T,3)

[0394] Vd＝VecADD32(T,Vd)

[0395] SHA256SU1Vd.4S,Vn.4S,Vm.4S

[0396] T0＜127:0＞＝Vm＜31:0＞:Vn＜127:32＞

[0397] T1＜63:0＞＝Vm＜127:64＞

[0398] T1＜63:0＞＝VecROR32(T1＜63:0＞,17)XOR VecROR32(T1＜63:0＞,19)XOR VecSHR32(T1＜63:0＞,10)

[0399] T3＜63:0＞＝VecADD32(Vd＜63:0＞,T0＜63:0＞)

[0400] T1＜63:0＞＝VecADD32(T3＜63:0＞,T1＜63:0＞)

[0401] T2＜63:0＞＝VecROR32(T1＜63:0＞,17)XOR VecROR32(T1＜63:0＞,19)XOR VecSHR32(T1＜63:0＞,10)

[0402] T3＜63:0＞＝VecADD32(Vd＜127:64＞,T0＜127:64＞)

[0403] Vd＝VecADD32(T3＜63:0＞,T2＜63:0＞):T1＜63:0＞

[0404] 指令假设循环队列常驻于四个4x32位向量寄存器中。指令不排除使用调度扩展。

[0405] 在指令内部实现元素的重新排序及提取。然后,微架构能够选择以将这些及固定移位与旋转实施为布线。

[0406] 在大部分微架构中,指令可具有低循环等待时间。

[0407] 因此,根据示例性实施例,单一指令多重数据处理电路经配置以由第一调度指令控制,该第一调度指令具有输入操作数Sp[127:0]且产生输出操作数Sq[127:0],该输出操作数的值与由以下步骤给出的值相同:

[0408] T[127:0]={Sp[31:0]:Sq[127:32]};

[0409] T[127:0]＝VecROR32(T[127:0],7)XOR VecROR32(T[127:0],18)XORVecROR32(T[127:0],3);及

[0410] Sq[127:0]=VecADD32(T[127:0],Sq[127:0]),

[0411] 其中VecROR32(A,B)为在A内的每个32位字单独右旋转B位位置,且VecADD32(A,B)为在A内的每个32位字单独增加至在B内的相应32位字。

[0412] 因此,根据示例性实施例,单一指令多重数据处理电路经配置以由第二调度指令控制,该第二调度指令具有第一输入操作数Sp[127:0]及第二输入操作数Sq[127:0]且产生输出操作数Sr[127:0],该输出操作数的值与由以下步骤给出的值相同:

[0413] T0[127:0]＝{Sq[31:0]:Sp[127:32]};

[0414] T1[63:0]=Sq[127:64];

[0415] T1[63:0]＝VecROR32(T1[63:0],17)XOR VecROR32(T1[63:0],19)XOR VecROR32(T1[63:0],10);

[0416] T3[63:0]＝VecADD32(Sr[63:0],T0[63:0]);

[0417] T1[63:0]=VecADD32(T3[63:0],T1[63:0]);

[0418] T2[63:0]＝VecROR32(T1[63:0],17)XOR VecROR32(T1[63:0],19)XOR VecROR32(T1[63:0],10);

[0419] T3[63:0]＝VecADD32(Sr[127:64],T0[127:64])；及

[0420] Sr[127:0]＝{VecADD32(T3[63:0],T2[63:0]):T1[63:0]}，

[0421] 其中VecROR32(A,B)为在A内的每个32位字单独右旋转B位位置，且VecADD32(A,B)为在A内的每个32位字单独增加至B内的相应32位字。

[0422] 在SHA-256与SHA-512之间的差异

[0423] SHA-512算法非常类似于SHA-256算法。在描述对SHA-256的支持的部分中概述的相同方法可同样地应用于SHA-512，仅存在以下小的不同：

[0424] 输入数据分成128字节的区块，且以大端(big endian)形式处理为16x64位字。

[0425] SHA-512对8x64位字有效

[0426] SHA-512需要散列函数的80次迭代。

[0427] 散列函数及调度更新对64位字有效，且散列函数及调度更新含有不同固定移位、旋转，及逻辑异或。

[0428] 为了简便起见，我们忽略SHA-512算法。

[0429] 将SHA-512、SHA-384、SHA-512/256及SHA-512/256算法作为目标的指令。

[0430] 将SHA-256作为目标的指令的目的与对于SHA-512算法的目的相同。我们列出将SHA-512作为目标的这些指令的可能实现。

[0431] 在SIMD寄存器为128位，且假定每散列4次迭代及调度指令的情况下：

[0432] SHA512H{Qd,Qd+1}，{Qn,Qn+1}，{Vm.2D,Vm+1.2D}

[0433] SHA512H2{Qd,Qd+1}，{Qn,Qn+1}，{Vm.2D,Vm+1.2D}

[0434] SHA512SU0{Vd.2D,Vd+1.2D}，{Vn.2D,Vn+1.2D}

[0435] SHA512SU1{Vd.2D,Vd+1.2D}，{Vn.2D,Vn+1.2D}，{Vm.2D,Vm+1.2D}

[0436] 请注意，上述指令有可能需要寄存器锁定(register pinning)；在微架构之内指定一个寄存器且暗示第二寄存器。指令将不再属于典型RISC三元形式，然而，这些类型操作存在优先，例如，在ARM有限公司的Neon载入/存储多重指令中。

[0437] 在较宽SIMD寄存器可用的情况下，指令的可能的变体包括：

[0438] SHA512H Qd,Qn,Vm.4D

[0439] SHA512H2Qd,Qn,Vm.4D

[0440] SHA512SU0Vd.4D,Vn.4D

[0441] SHA512SU1Vd.4D,Vn.4D,Vm.4D

[0442] 这些指令的变体亦处理用于散列的迭代及调度更新操作，但是归因于较宽SIMD寄存器符合三元RISC形式。

[0443] 使用如FIPS180-4中所述的截断，这些指令可同样地以SHA-384、SHA-512/256及SHA-512/224为目标。

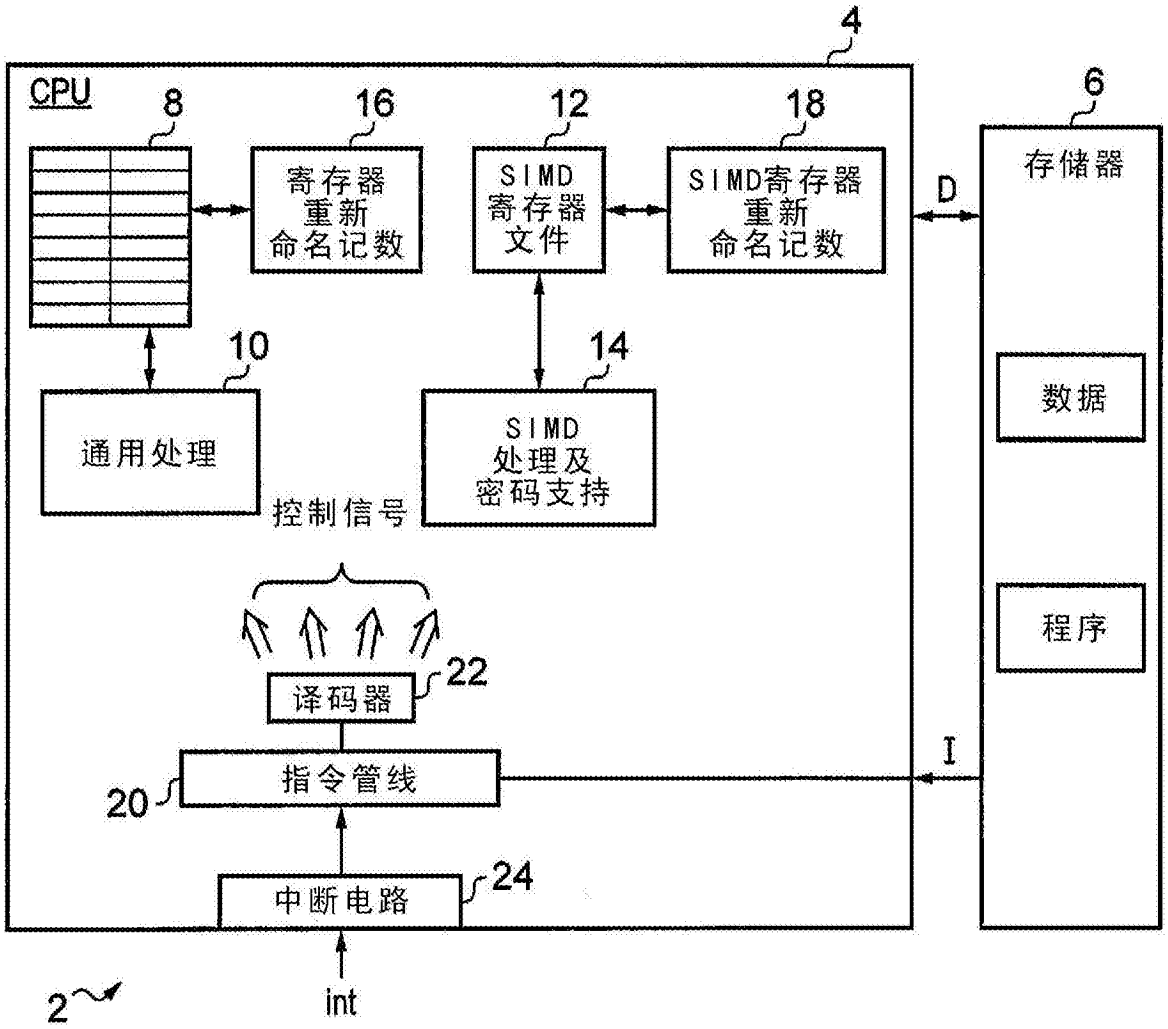

4
CPU
8
16
12
18
6
寄存器
重新
命名记数
SIMD
寄存器
文件
SIMD寄存器
重新
命名记数
D
存储器
10
14
通用处理
SIMD
处理及
密码支持
数据
控制信号
程序
译码器
22
I
20
指令管线
中断电路
24
2
int

图1

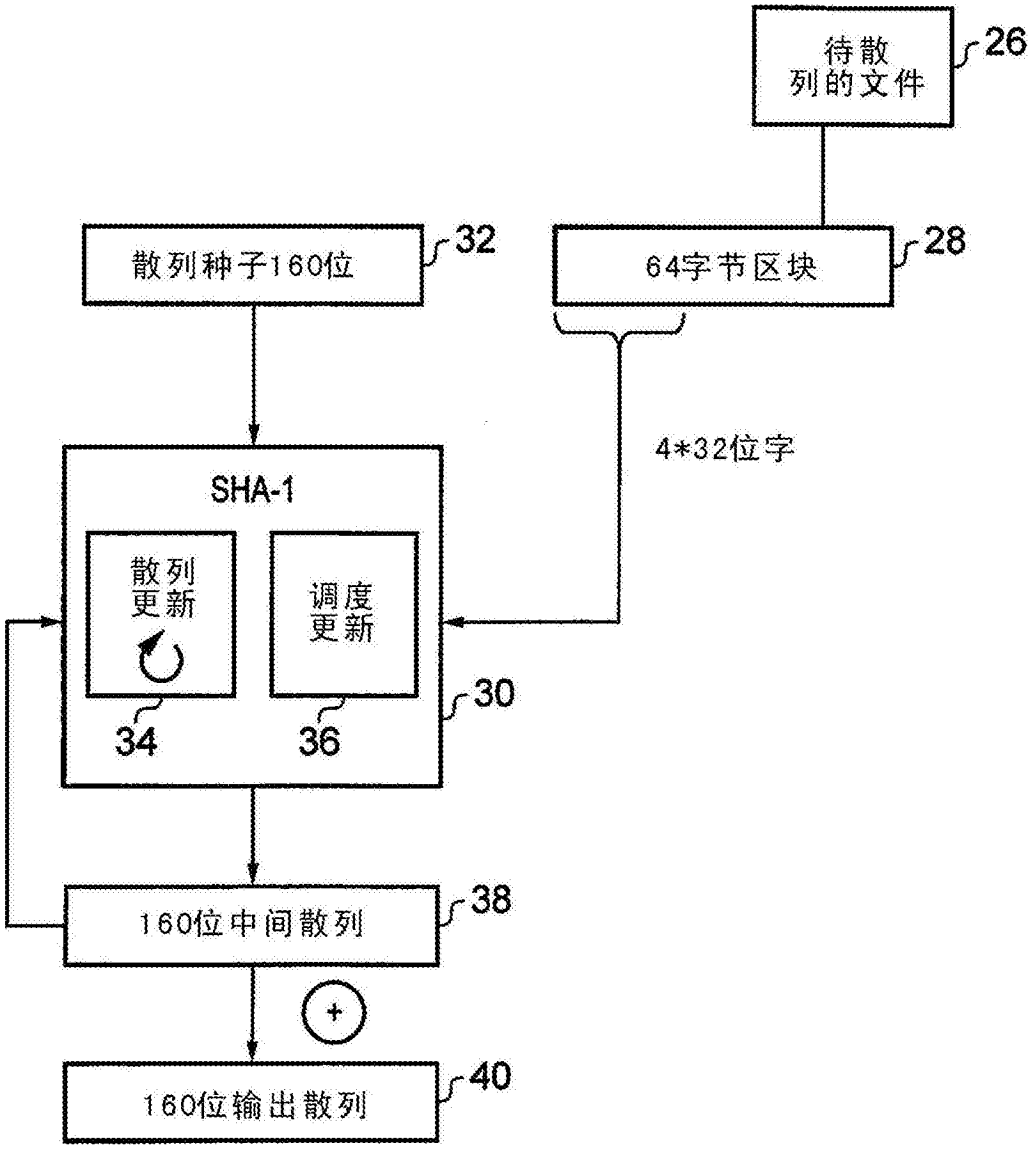

待散列的文件
26
散列种子160位
32
64字节区块
28
4*32位字
SHA-1
散列更新
调度更新
34
36
30
160位中间散列
38
+
160位输出散列
40

图2

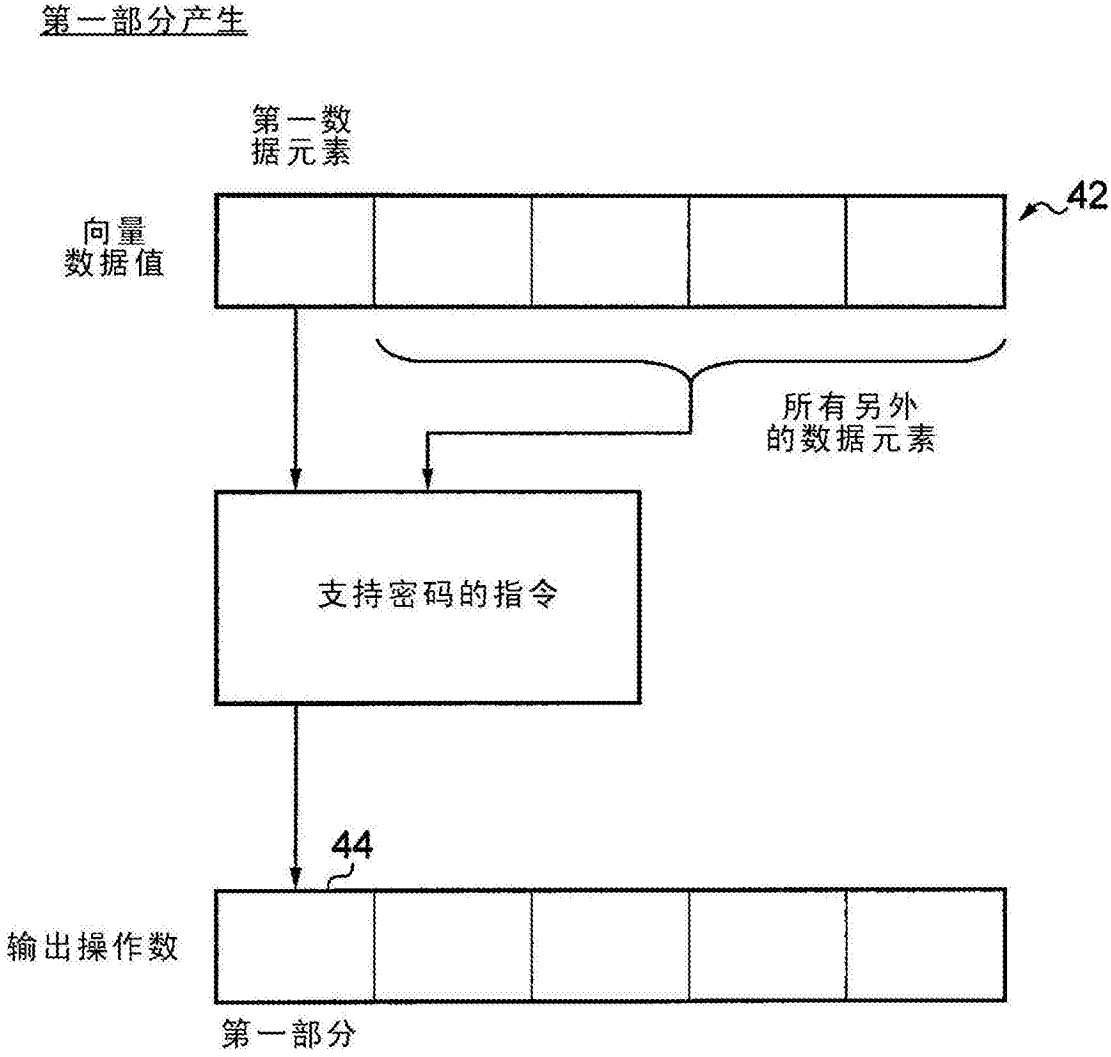


图3